中国科学院大学课程讲义

数据科学 R 与 Python 实践

中国科学院大学 经济管理学院

2022 年 1 月 1 日

前 言

随着大数据国家战略出台，中央要求各级领导干部多一个基本功：增强利用数据推进各项工作的本领。2017 年 12 月 10 日,习近平总书记在中央政治局第二次集体学习时的重要讲话指出：“...现在，世界各国都把推进经济数字化作为实现创新发展的重要动能，在技术研发、数据共享、安全保护等方面进行前瞻性布局。...善于获取数据、分析数据、运用数据，是领导干部做好工作的基本功。懂得大数据，用好大数据，增强利用数据推进各项工作的本领，已经成为领导干部的新时代必修课。...”如今数字经济和科技事业已经走向可预见的未来了。

科学数据已成为科技发展的“头等公民”（Research Data as First-Class citizen）。当前，数据科学家、数据公民、数据管理员正在成为创新社会的新兴动力。目前环绕数据科学、数据政策、数据管理的课题日益重要，不仅社会各界都已面临实际发展战略上的这个核心问题，而且也将直接影响个人职业生涯的生存与未来发展。当此之时，至此之际，这门课程需要兼容国际国内的青年科研人才的需求。

本课程旨在通过案例数据的 R 与 Python 的上机操作，辅之讲解必要的知识理论，以及介绍运用在科学研究和企业服务的经验，以“从做中学”的方式进行授课。课程秉持“不谈颠覆、不侃概念、只做实践”的原则，环绕六项指示精神“技术研发、数据共享、安全保护、获取数据、分析数据、运用数据”等进行设计，上课同学把具体科研课题与授课老师进行交流，往往能够取得更好的学习成效。这份教材仅仅是个开端，抛砖引玉，立足信息素养，发展更多新的内容。

参考资料很多，包括但不限于：

（1）朝乐门. 数据科学[M].清华大学出版社,2016.
（2）Mailund T.R Data Science Quick Reference[M]. Springer, 2019.
（3）Kenett R.S. & Redman T.C. The Real Work of Data Science[M]. Wiley, 2019.
（4）Cuadrado-Gallego J.J. & Demchenko Y. The Data Science Framework[M]. Springer, 2020.
（5）Rahman A. Statistics for Data Science and Policy Analysis[M]. Springer, 2020.
（6）Blum A..Hopcroft J..Kannan E. Foundations of Data Science[M]. Cambridge University Press, 2020.
（7）Fan J..Li R..Zhang CH. et al.Statistical Foundations of Data Science[M]. Taylor & Francise Group, 2020.
（8）Emmert-Streib F..MoutariS..Dehmer M. Mathematical Foundations of Data Science Using R[M]. De Gruyte, 2020.
（9）Rautaray S.S..Pemmaraju P..Mohanty H. Trends of Data Science and Applications Theory and Practices[M]. Springer, 2021.
（10）Balas V.E..Hassanien A.E..Chakrabarti S. et al. Proceedings of International Conference on Computational Intelligence.Data Science and Cloud Computing[M]. Springer, 2021.
（11）Tollefson M.Visualizing Data in R4[M]. Apress, 2022.

目　录

第 1 章　数据科学绪论......6
第一节 数据世界与真实世界......9
一、哲学思维......9
二、科学思维......9
三、工程思维......10
第二节 数据科学的思路、方法、流程......12
一、数据科学的思路......12
二、数据科学的方法......12
三、数据科学的流程......14
第三节 开放数据的获取、使用与传播......15
一、开放数据的获取......15
二、开放数据的使用......16
三、开放数据的传播......17
第四节 开源软件的获取、使用与传播......17
一、开源软件的获取......18
二、开源软件的使用......19
三、开源软件的传播......19
第五节 数据科学入门途径......21
一、体系化的学习......21
二、个体化实践......22
本章小结......23
习　　题......24
第二章　数据处理篇......26
第一节 创建数据表格......29
一、创建数据......29
二、用 R 创建数据表格......30
三、用 Python 创建数据表格......31
第二节 存取数据表格......33
一、数据的存储与取用......33
二、用 R 存取数据表格......33
三、用 Python 存取数据表格......34
第三节 读写数据集......36
一、数据读写......36
二、用 R 读写数据集......36
三、用 Python 读写数据表格......37
第四节 更改文件格式......39
一、数据文件的更改......39
二、用 R 更改文件格式......39
三、用 Python 更改文件格式......41
第五节 撰写脚本语言并且保存......43
一、撰写脚本语言......43

二、用 R 撰写脚本语言......43
三、用 Python 撰写脚本语言......43
第六节 修改脚本语言并且执行......44
一、修改脚本语言......44
二、用 R 执行脚本语言......44
三、用 Python 执行脚本语言......45
本章小结......47
习　　题......47
第三章　数据型塑篇......52
第一节 数据检查......55
一、系统检测......55
二、人员检测......58
第二节 数据抽取......61
一、数值替换......61
二、行列选择......62
第三节 数据合并......65
一、利用矩阵的方法......65
二、利用索引的方法......66
本章小结......67
习　　题......67
第四章　统计分析篇......72
第一节 分布状态......76
一、描述统计......76
二、单样本 T 检验......78
第二节 变量关系......83
一、皮尔森相关系数......83
二、斯皮尔曼等级相关系数......83
三、一元回归方程......85
第三节 抽样方式......89
一、重复随机抽样......89
二、不重复随机抽样......89
三、随机森林......91
本章小结......95
习　　题......95
第五章　数据建模篇......97
第一节 建模......98
一、基本概念......98
二、误解误用......99
三、模型选择......100
四、实施要点......102
第二节 排名......105
一、社会网络......105
二、排名算法......112
三、算法实现......117

四、算法优化......120
第三节 推荐......123
一、推荐算法......123
二、分布计算......130
三、词频计算......133
四、相似推荐......139
本章小结......144
习　　题......144
第六章　机器学习篇......146
第一节 基本原理 147
一、发展简史......147
二、目的任务......148
三、应用原则......153
四、项目练习......154
第二节 无监督学习......162
一、主成分分析......162
二、因子分析......168
三、因子分析与主成分分析的综合用法......171
四、关联分析......173
第三节 有监督学习......176
一、贝叶斯学习......176
二、最近邻学习......188
三、决策树......195
四、随机森林决策树......205
本章小结......211
习　　题......211
第七章　深度学习篇......213
第一节 基本原理......214
一、发展历史......214
二、关键问题......219
三、解决方案......226
四、软件操作......232
第二节 基础知识......243
一、最小二乘法 OLS 回归......243
二、罗吉斯回归......246
三、神经网络......256
四、多隐藏层神经网络......263
第三节 深度学习......268
一、深层神经网络......268
二、卷积神经网络......272
三、更深层卷积神经网络......282
四、抽取权重与模型评估......290
本章小结......295
习　　题......295

结 语..297

第一章　数据科学绪论

学习目标

初识数据世界与真实世界的区别；
初识数据科学的思路、方法和流程；
了解开放数据的获取、使用和传播；
了解开源软件的获取、使用和传播；
理解如何迅速掌握数据科学的途径。

知识结构

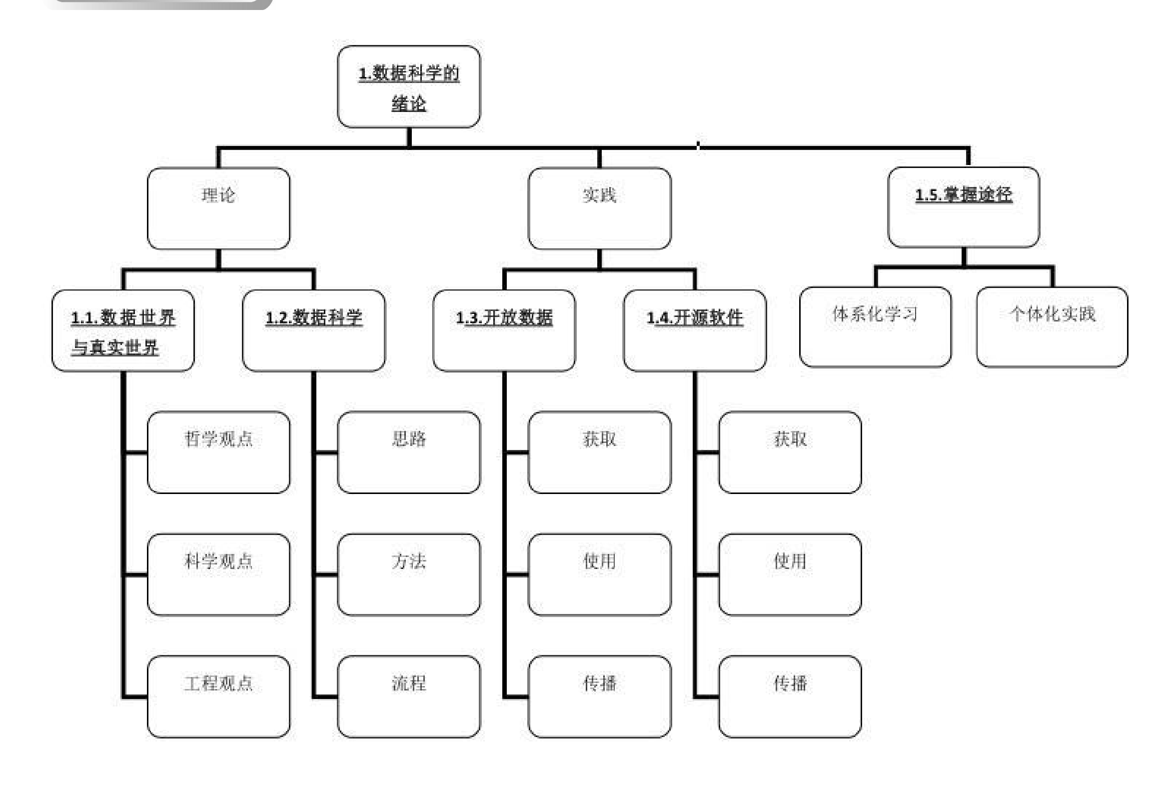


图 1.1 数据世界与真实世界的概述图



导入案例

开始接触数据科学会产生的困扰。

如果您在图书馆、书店以及互联网上，寻找与数据科学有关的图书、论文、以及介绍短文，会发现有各种各样的情况。在一方面，您可能找到数学公式的（如拓扑学或者运筹学）、数据库的（如 Orcle 或者 SQL）、脚本语言（如 Python 或者 R Language）的、商用软件（如 Excel）的、数据分析的（包括软件工具使用和统计解读），甚至商业战略等（如 MBA 课程）不同角度出发讨论数据科学的内容；另一方面，您也可能接触到著作以外的软件、代码、教学数据、课件、课程视频、短视频、商业广告、线上课程、线

上互动平台、小程序等不同数字信息资源，以数据科学为名的实验素材。如果您仔细翻查字典，在网上搜索维基百科或者某些权威机构的官方网站上的词条，试图找寻数据科学的准确定义，则会看到更多专有名词来解释数据科学这个专有名词。如何有效掌握这门知识、技术、技能以及核心思维方式？目前，我们提供的方式，就显得很有必要找到一条适合入门水平程度的道路。一是提纲挈领，把重要的内容，按照章节次序进行排列，有序不乱地建构一个框架，提升认知层面的记忆能力；二是从做中学，把授课内容、学习任务、课后习题，与自身面临的研究问题结合起来，提升感受层面的领会能力。

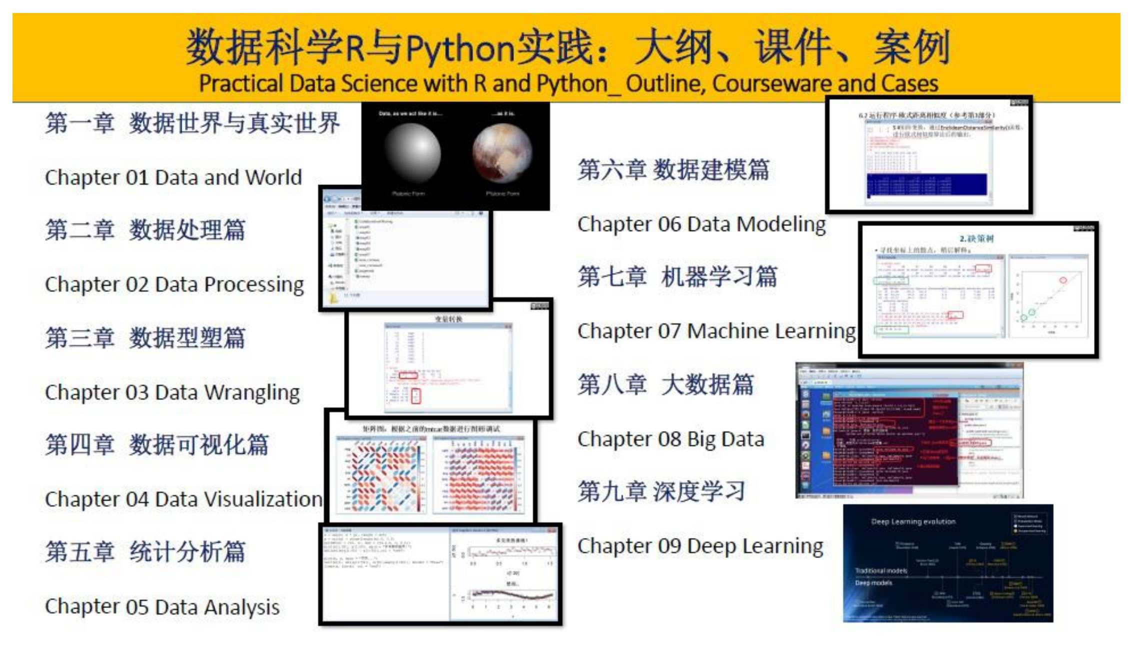


图 1.2 数据科学 R 与 Python 实践的课程体系

目标不同，途径不同。接触《数据科学 R 与 Python 实践》的各位，可能通过课程达到以下效果：.

（1）具备数据科学的系统性、体系化、结构化的知识。

（2）掌握 R Language 以及/或者 Python 的基础技能。

（3）寻求进阶的数据科学、R language、Python 的实践能力。

（4）解决具体的研究问题、项目课题、数据。

根据以往教学相长的经验，在讲授这门课程时，或者在家自学，需要事先做好心理建设，再全力以赴。请您先考虑以下几点：

（1）所谓“快乐的”学习是不存在的，仅仅在于初期刚刚掌握一门专业梗概之后，对于未来过于乐观预期的错觉。

（2）所谓“痛苦的”学习也是不存在的，因为人们会本能地离开或者说是逃避那些不愉快的事务，停止痛苦意味着停止学习。

（3）所谓“快乐并且痛苦着”才是进入状况的常态。这时候“老师”和“临摹”就无疑是关键。

因为工作需要，也因为个人生活习惯，我总是在积极学习新的事务，可能是当时对所有人都是新的事务，也可能是对一般人而言很普通但是我却很困难很费解才能掌握，因为这样，特别请教了一些具有教学经验的老师，如何走出学习低谷，之后归纳如下：

（1）首先描绘“专业知识技能的学习曲线”。

（2）熟悉“所用资源和所需知识”。

（3）确定每个阶段的教学、学习或者自学重点 。

如果研究员或者大学教授在科研院所或者高等学校讲授这门课程，建议如下安排：

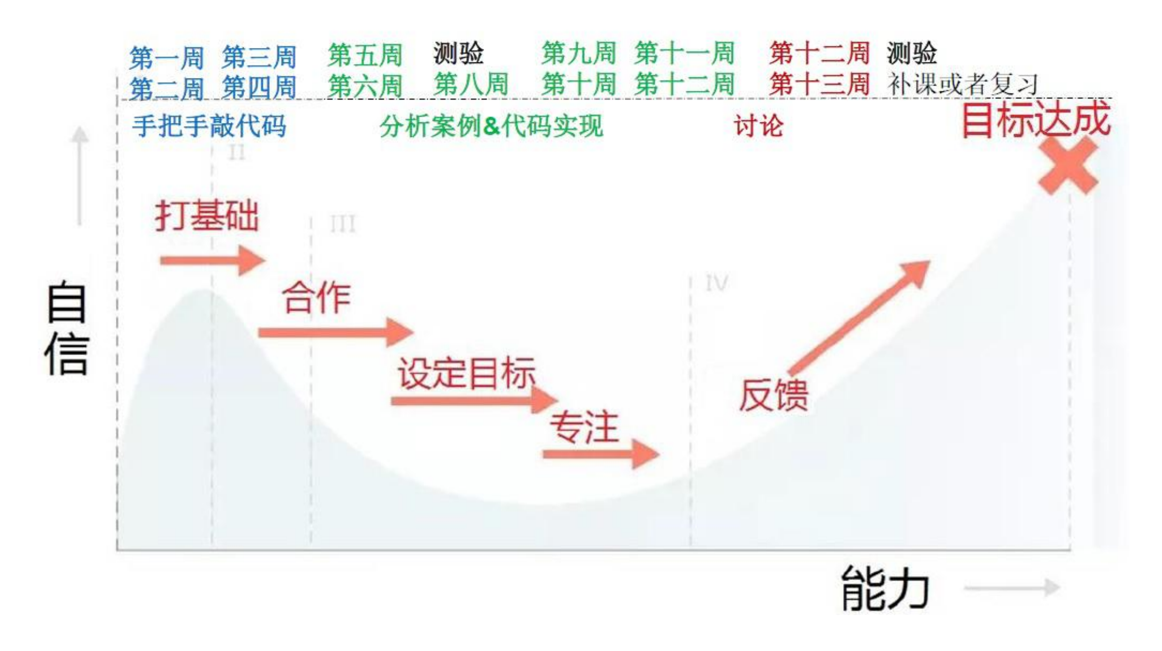


图 1.3 数据科学 R 与 Python 实践的课时建议

我们所学习和投资的东西，应该是能够积累的知识，这样知识就可以建立在已有知识的基础上，不断迭代更新。因此，与其学习一些明天就可能过时的东西，比如某种特定类型的软件使用方法（在两年后甚至没有人使用），不如选择那些能让您在 10 年或 20 年后变得更聪明的东西。所以，现在让我们一起通过 R Language 和 Python 来掌握数据科学！

第一节 数据世界与真实世界

一、哲学思维

我们并非人人想要成为哲学家，但是通过抽象一点高层思考，可以帮助我们更好的概括数据科学，特别是，在进行数据科学 R 与 Python 实践之前，能够清楚划分它的边界。

数据世界和真实世界存在距离。我们不能完全理解真实世界，甚至应该进一步要深思我们如何理解世界以及我们如何感受世界，这是哲学问题。对于学术理论而言，它在数据世界和真实世界之间，通过语言（文字、图像、计算机的各种表达形式）建立了数据世界和真实世界的桥梁。虽然我们不能真正意义上的知道什么是真实世界，也很难穷尽数据世界的每个细节（细粒度的无穷边界），但是通过人为制造的学术理论，能够让我们进行探索，充分运用起我们的智力和感情。

哲学思维，强调我们不一定能够真正认识到真实世界，不一定能够真正制造出数据世界，但是通过学术理论能够让我们感受到和相信有真实世界的存在。因此，如果已经掌握了一定的知识，无论是什么领域，就有必要认识到学术理论建立在数据世界这个证据链上，但是学术理论和真实世界之间总是存在差距。有些模型、有些论文、有些著作，倾向于把世界过于理想化、简单化、容易记忆观点化，使得人们容易把人为的数据世界（并不完全虚幻）当成真实世界。

（一）科学思维

我们并非人人想要成为科学家，但是掌握数据科学最重要的一个关键，就是首先具备科学思维的能力。

科学研究的核心在于变量之间的关系，把能够观测到的事物予以计算，形成简化后的模型，来解释甚至预测世界。

所形成的简化模型，就是数据世界；所反映的是真实世界里，各个要素的紧密关系。如果我们不讨论变量之间的关系，就无从讨论任何有深度的内容。所以，数据世界，需要回答关于变量的提问：

1.这两个变量是否有关系？

2.如果有，它们的关系是否显著？

3.这些关系是什么关系，能否用数学模型来描述？

4.这个关系是否带有普遍性？

5.这个关系是不是因果关系？

科学研究通过严格的逻辑定义问题，严格定义问题之后，是不断的自我否定。求解的过程就是提出解决方案，再自我否定，经过多次以及其他人员的帮助，达到无法继续自我否定的状态，才是发表暂时性结论的时候。

科学思维，强调：

1.理论适用与否靠实验或观测，而不是辩论、信仰或者权威。

2.基于含糊不清或者不适当的前提假设，所做的推理没有意义。

3.我们需要花费时间在那些依靠证据或数据，来定义问题以及具有准确概念含义的问题，而不是花费大量时间在背诵和牢记教条。

科学思维，注重验证，没有验证不是科学。在数据科学里，通常不只是一次检验，而是

多次、多方、多番检验：

1.人们鼓励利用所有可能的检验方法来找出问题；但永远不能证明绝对没有问题。

2.不应为了达到“接受零假设”的目的，采用多种检验统计量来检验，如果不能拒绝的多于可以拒绝的检验，则“接受零假设”；甚至把经典检验和非参数检验混起来使用，但是零假设和备选假设的含义在非参数情况下意义和经典情况并不相同，不能比较。

3.适当的做法是：即使仅有一个检验拒绝零假设就应该拒绝零假设，如果都不能拒绝，就不拒绝零假设（不能说接受）。

（二） 工程思维

我们并非人人想要成为工程师，但是如果从事数据科学的相关工作，则必须通过工程手段予以实现，才能凸显它的实用价值。

理论，即使没有数据作为支撑，也能指导我们收集数据；通过理论的拆解和细化，能够形成一系列的指标。

1.当拥有来自一些变量（指标）的数据或记录，但缺乏描述这些变量之间关系时，可以建立模型。

2.在有了一定的模型后，通过科学研究方法，可以确定数据是否令人信服地支持某种论点。

由此，我们可以通过工程实践，建立两种世界，即：基于理论、数据、模型的数据世界，以及基于数据世界所能反映的真实世界。

从数据科学的角度而言，我们更加倾向数据驱动，而不是模型驱动，来为我们建立这两个相同的世界。

模型驱动带来的负面影响：

1.套用模型，只看到有限方法和数据；

2.套用数据，快速得到图形；

3.快速发表论文；

4.文献堆积如山，比较难以查证可靠性和有用性，从而难以产生具有实际价值用途的研究。

数据驱动、问题驱动、用户驱动的形象化方式：

1.面对问题收集数据；

2.根据数据建立模型；

3.利用模型做预测或者得到结论；

4.针对模型产生的新的信息进行更新、判断、解读。

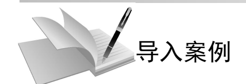
工程思维，强调我们能够建设数据世界，应用数据世界来反应和改造真实世界。因此，按照这个思维，数据科学的核心在于模型求解：

1.模型的解可能是近似模型的精确解(有较好数学背景的人的强项)；

2.也可能是近似模型的近似解（有较好统计背景的人的优势）；

3.这些解可能有在一定概念下的：“最优性”（需要专业背景）；

4.然而，越是精确，就越容易无视可能存在的真实现象，因此需要主动改造真实世界，在改造中证明它的正确性。



导入案例

人的行为，具有一定的随机性（randomness），但是在某个时间段里，以及在某个场景里，他们的

个人或者集体行为，是相对稳定。人的寿命，具有一定随机性。这种随机性可能和人的经历、基因、习惯等无数说不清的因素都有关系。但是总体而言，一个国家或者地区的平均寿命相对稳定。世界是随机性事件的集合，因此，随机之中有规律性，科研人员要做的是找到短暂改变人类命运的规律性。

一个网络信息服务的负载量是一组固定的数字，称为常数（constant）或者常量。但是，哪些用户多少次访问这个服务，就不一定了，这有随机性。行为特征和访问行为就是两组变量（variables），根据这两组变量，我们可以利用起数据，并且建立模型和检验。

第二节 数据科学的思路、方法、流程

一、数据科学的思路

首先，从数据看模型。数据，是关于变量的观测值。例如，有人通过社会科学调查方法，通过调查产生一些数字，这些数字就是数据（data）。变量只有用数量来描述时，才有可能建立数学模型，并使用计算机来分析。数据中它们通常用虚拟变量（dummy variable）代表，比如性别用 0、1 代表，三种收入用 0、1、2 代表(或用字母代表）。我们通过数据可验证有关的理论或假定。

第二，从变量看数据。当变量按照随机规律所取的值是数量时，该变量称为定量变量或数量变量（quantitative variable）；如果是随机的，称为随机变量（random variable)，例如购买某项商品或者服务的人数等。并非只有流量数据才有研究人类行为的价值，通过社会科学调查方法产生的数据，一样具有说服力，因为：（1）科研人员参与数据采集，除非刻意造假，否则科研人员对于数据产生的背景、用途、可能存在的问题，以及如何利用和数据能够和不能反映的现象，都有比较深刻的了解。（2）科研人员是从问题、采集数据、建模、验证等流程开始，而不是得到数据后仅用分析工具进行数据分析的象牙塔式工作。

第三，从问题看变量。我们需要理解数据产生的背景，而不是找到数据就能解释，因为：

（一）可能：数据反映的不是变化或者规律，而是采集数据的方法不同或者途径不同所产生的不同视角下的记录。

（二）可能：数据反映的是长期现象中本来没有的一个突发现象。

（三）可能：数据所反映的显著性仅仅是理论上的显著性，而不是实际工作业务上的现象产生了某个值得关注的现象。

最后，从模型看理论。在一方面，在行业尊重科学研究并且形成较好的交流机制的前提下，通过深入探索数据集（即：数据世界）的内在规律，进行理论验证或者探索出不一样的结果，能够重复结果，并且总结出新的解释，形成创新。另一方面，从模型（检验）看理论，则是尽量避免理论误导，包括：（1）低级误导：有意无意地利用某些图像和文字，随意解读数据和信息。（2）高级误导：利用数据及统计方法上的选择来达到自己的目的。

二、数据科学的方法

质量差的数据得不到高质量的结论。好的方法不如好的数据。

第一步工作是数据采集。

（一）据实际目的进行收集：有些是人工采集，例如：问卷调查、访谈调查、观察记录、实验室采集等手段获得；有些是通过信息系统，比如：日志数据、查询数据、系统日志、用户日志、查询输入数据等。

（二）确定哪些变量的数据需要收集，这与数学、应用统计或者信息系统架构无关，而是对行业领域的了解程度和经验。 再者，不是有了数据，就要马上得到所需要的结论，而是取得：与所关心的问题直接有关的变量的数据。

第二步工作是数据处理。

原始数据往往或多或少地存在各种缺失值，以及不合逻辑或不一致等的情况，这就需要预处理方案。这类工作很可能耗时而且琐碎，但必须完成，否则无法后续分析或者造成的负

面影响难以估量。处理缺失值的方法，包括：

（一）整条数据及其所有变量的数据均删除；

（二）用同一变量其他值的均值或中位数填补；

（三）在各个变量之间建立模型（比如回归模型、最近邻方法、随机森里等）进行填补。

第三步工作是模型选择。

模型的目的是：预测、解释，或者理解产生数据的机制。科研人员的直觉是很重要的能力，也是判断研究能力的一种展现，科学研究的“工业流水线”很容易学习和掌握，但直觉则是需要长期锻炼和积累。我们通常进行探索性分析，或者寻找已有理论模型，两种方式。

探索性分析，包括：（一）利用图形；（二）描述统计；（三）探索分析方法，如：关联性、线性、异方差性、多重共线性、聚类特征、平衡特征、分布形状等。

寻找现成的模型，包括：（一）比较各种模型的计算结果；（二）如果现有模型不能满足需要，就可能产生新的分析方法；（三）模型选择的过程贯穿于整个数据分析过程。


阅读材料

假定

假定，是源于信仰或者方便；假定的选择和科研人员的直觉有关。（我们这里指的不是随意凭感觉，而是凭经验，拍脑袋也有其道德高标准，包括何时应该使用何时应该慎用这种能力的选择，也是直觉的一部分）。传统上，进行应用统计时，通常需要假定分布（如正态性）和模型，并且决定损失函数，由此制定检验值和临界值的判别准则。假定，它无法用确定性方法被验证，仅能尝试采用显著性检验来拒绝假定的正反两面，因此把假定换个词叫假设；然而，即使没有充分理由否定反面，也不能“证明”正面是正确，它仍然属于假定。

第四步工作是交叉验证（cross validation）。

算法模型，由于没有传统模型的那些假定，所以 判断模型优劣通常仅用交叉验证。交叉验证适用于传统模型之间，或者在传统模型和算法模型之间的比较：

（一）拿一部分数据作为训练集（training set）得到模型；

（二）再用另一部分数据（称为测试集 testing set）来看误差；

（三）有时需要进行 k 折交叉验证（k-fold cross validation）把数据分成 k 份，每次拿 k-1 份作为训练集，用剩下的一份作为测试集，重复 k 次，得到 k 个误差作出平均，以避免仅用一个测试集可能出现的偏差。


阅读材料

误差

误差越小，模型越优，建模意义越大，学说的可靠性越强。

误差，有两部分组成：

（1）研究结果在假定的模型之下的精确度，包括：相合性、无偏性等。

（2）假定的模型和真实的规律之间的距离。

如果是问题驱动、数据驱动的科研人员，就能够回答上述两个问题，但是如果是以模型驱动的科研人员只能选择回避这个问题。

第五步工作是解释模型。

选择模型不是最终目的，最终目的是解释模型所产生的结果；

（一）结果必须是应用领域的结果，需要具有实际意义；

（二）仅仅用统计术语或者计算机术语说某个模型较好，缺乏行业知识和判断，不是可靠的模型选择。

至此，请迅速回想或者翻阅本章第一节和第二节前部分内容，进行片刻思考。

三、数据科学的流程

我们反复强调 1.1.2 科学思维的重要性，并且认为“工业流水线”相对容易掌握，而“辩证地看待每个阶段的结果”却不容易。为了不让大家不知所措，我们此处引用一副著名的流程图，来说明 1.1.3 的工程思维，如果在课题组要如何实现的方式。

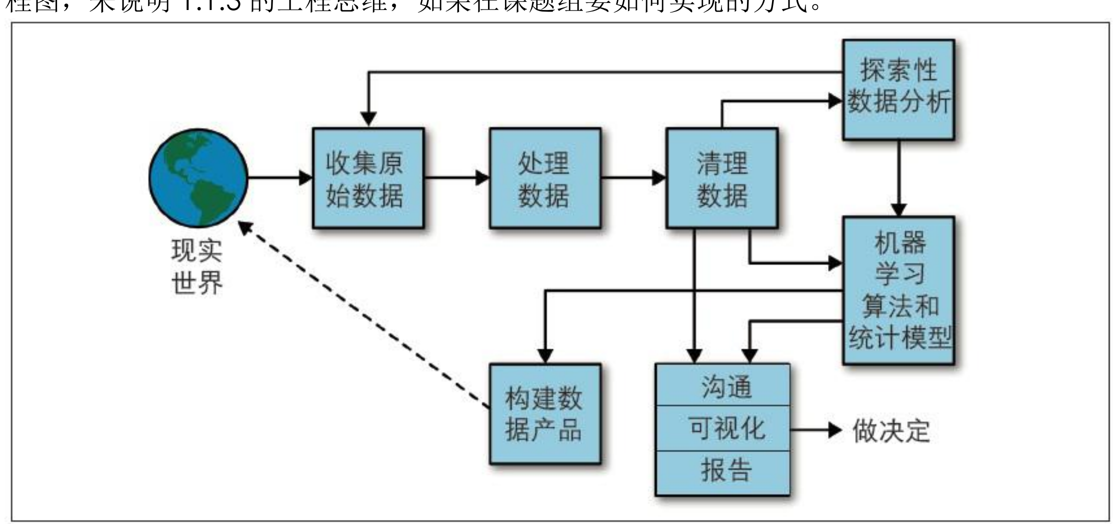


图 1.4 数据科学的工作流程（取自《数据科学实践》一书）

请根据图 1.4 对照您的工作或者所见过的数据产品、数据分析、数据处理工作等，进行思考。此处不再花费篇幅逐一讲解介绍。

第三节 开放数据的获取、使用与传播

一、开放数据的获取

开放并非意味着没有知识产权，与之相反，只有在具备知识产权的情况下，数据拥有者才能够决定是否开放共享或者不作开放共享。

通常情况下，作者或者著作权持有者，如果愿意分享他所拥有的数据集，那么他就会通过开放许可协议，把他的数据放到互联网上给人们使用。人们拿到这些数据集之后，需要注意开放许可协议，是否限制了他们进行商业用途或者其他方面的限制。

通常情况下，人们使用他人的数据集，需要引注来源，并且说明做了哪些数据处理。

上述语言可能对大家来说，过于拗口，但是已经是非常简化的描述了，需要大家仔细思考和记住。为了让大家理解此事的重要性，我们举出一个案例。

案例介绍

鸢尾花 iris 数据集，经常为教学使用，出现在许多机器学习的教学场合里。按理说，应该是经过多次试验成功的一组数据集，与之配套的机器学习算法，无论是 R 还是 Python 都有很多。然而，实际上，人们在网上获得 R 或者 Python 的代码之后，以及获得 iris 数据集之后，却往往不能重复教科书或者课程视频里的实验结果。

原因很多，其中之一，是网上出现许多不同的 iris 数据集，因为它们的数据结构、索引方式和行列有所不同，导致我们不能简单套用已经做好了的示例代码。我们需要经过数据检查、处理和清晰之后，略微调整代码，才能进行数据分析，得到结果。

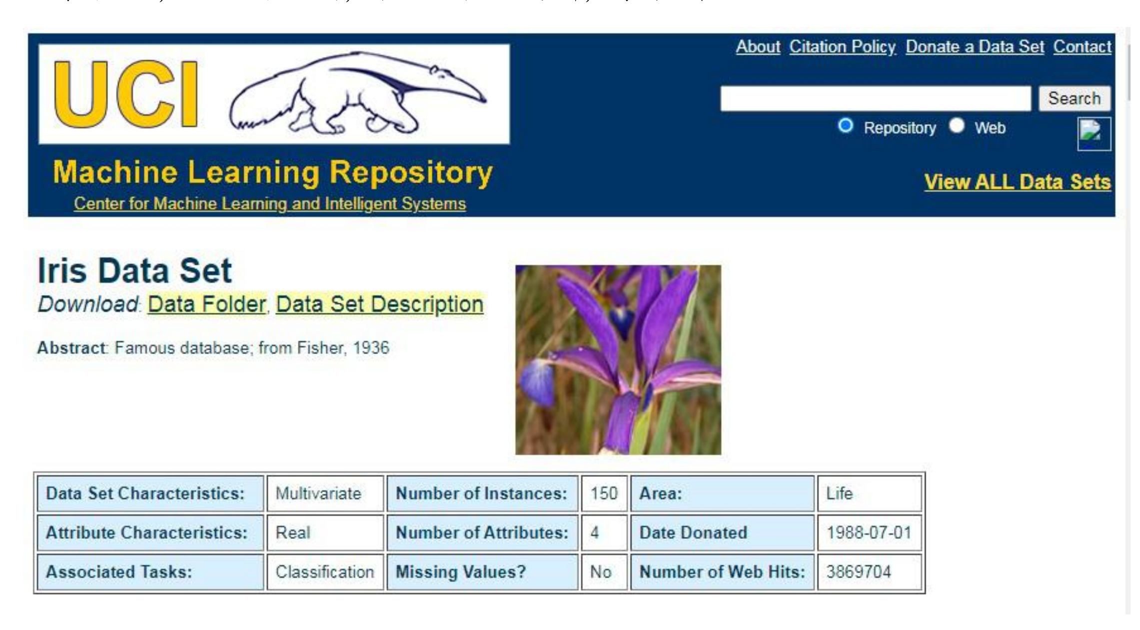


图 1.5 鸢尾花 iris 数据集的来源

我们在 R 软件里，可以找到 R 自带的数据集，其中就有 iris 数据集，但是它是经过修改后的数据集，而非原始发表的那个 iris 数据集；我们经常看到的 python 代码，所处理的 iris 数据集，也是面向不同版本的 iris 数据集，有些甚至刻意经过缺失值处理，为了方便教师指导学生们练习

如何处理缺失值，因此代码及其附加文件（iris 数据集），也与原厂不同。

按理来说，人们应当给于参考出处：来自 UCI Machine Learning Repository 的 Cancer 数据集，进行演示和讨论（http://archive.ics.uci.edu/ml/datasets/Iris）。

二、开放数据的使用

我们仍然以 Iris 数据集为例；当我们准备开始使用开放数据，除了找到原始的、准确的、完整的出处信息以及获得最为可靠的数据集版本，在使用时务必注意：

（一）数据来源：确定数据贡献者，有名有姓，可供人们查证的信息，其所提供的数据或者信息相对可靠。

（二）数据集信息：多少行、多少列、数据格式、数据量...等的信息，至关重要。因为我们拿到一组数据集之后（在第二章详细介绍），必须马上进行检查。此时数据集的信息有助于我们进行核对，是否拿到真实可靠的那个数据集版本。

（三）变量信息：数据集的每个变量名称，各自代表什么意思，这有助于我们运用领域知识对于分析和检验结果做出判断。

（四）相关论文：帮助我们了解数据集的产生背景、处理方法、使用条件等一系列在我们拿到数据集之前所发生的事情，即：该 Iris 数据集的历史。

（五）引用该数据集的论文：有助于我们知道别人曾经怎么利用这个数据集，进行了哪些研究工作，我们可以模仿学习或者做出不一样的成果（在 1.5.2 做说明）。

案例介绍

鸢尾花 iris 数据集，在我们使用之前，得要看清楚它的父母（贡献者）、长相（数据大小）、本质（数据内容）、历史（数据怎么来的）和其他人的用法。

Creator:

R.A. Fisher

Donor:

Michael Marshall (MARSHALL%PLU '@' io.arc.nasa.gov)

Data Set Information:

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Predicted attribute: class of iris plant.

This is an exceedingly simple domain.

This data differs from the data presented in Fishers article (identified by Steve Chadwick, spchadwick '@' espeedaz.net). The 35th sample should be: 4.9,3.1,1.5,0.2,"Iris-setosa" where the error is in the fourth feature. The 38th sample: 4.9,3.6,1.4,0.1,"Iris-setosa" where the errors are in the second and third features.

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

Relevant Papers:

Fisher,R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
[Web Link]

Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
[Web Link]

图 1.6 鸢尾花 iris 数据集的信息

我们在 R 软件所找到的 Iris 数据集，也有上述说明，表明了它与 UCI 的 Iris 的差别。一

般人之所以用错，是因为没有注意开放数据的使用细节，或者缺乏知识产权意识。

三、开放数据的传播

当我们在实验室从事一系列的科研工作之后，会产生我们自己的科研数据。涉及到科学数据、科学大数据、科研数据、数据监管、数据管理计划等的内容，不是我们这里所要讨论的重点。限于篇幅，此处我们用图 1.7 表示开放数据的流程图。

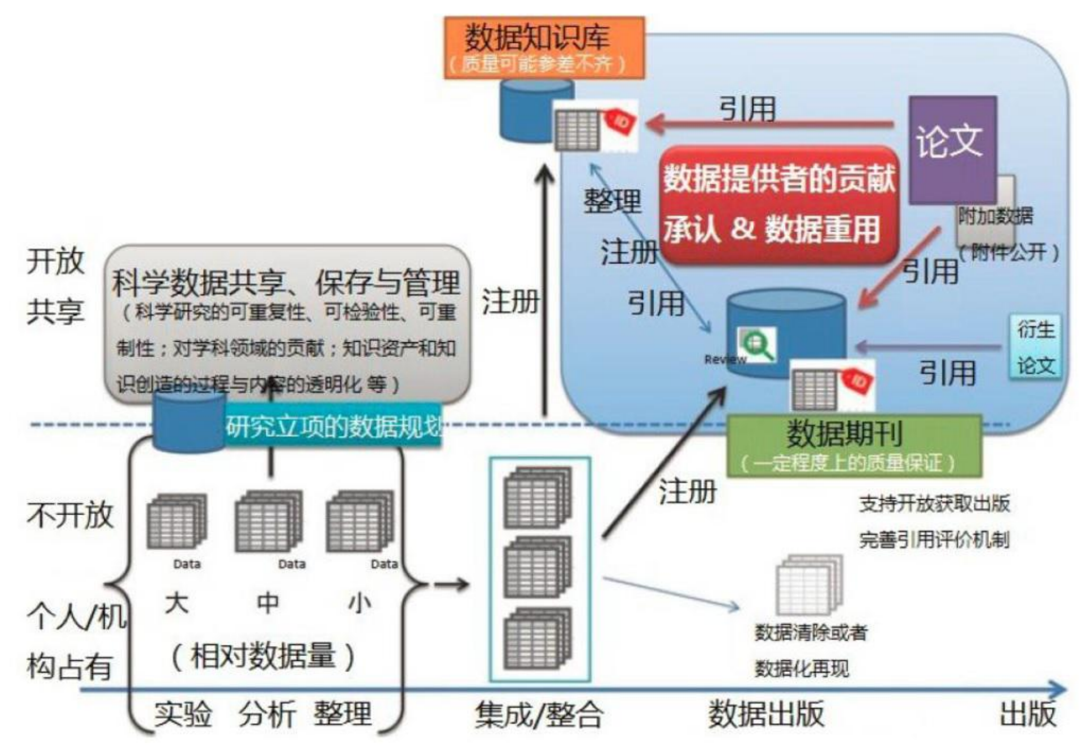


图 1.7 开放数据的流程图

如前所述，唯有在课题组愿意数据提供给众人的时候，才会选择一系列的注册途径，把数据集存储在数据知识库，提供想要使用的人们下载，同时有可能会把前面提到的数据集的历史和来源，做一番交代，写成数据论文进行发表。这样，其他人在使用课题组的这份数据集的时候，也可以通过引用课题组发表的数据论文，达到承认课题组的贡献与帮助，以及说明它们使用的是否为真实可靠的那份数据集。

第四节 开源软件的获取、使用与传播

一、开源软件的获取

开源软件运动，早于科技论文开放获取以及科学数据开放共享（第三节提到的开放数据的形式之一），我们此处不介绍开源运动。限于篇幅，我们提醒大家注意开源软件的几个情况。

首先，通过对比方式，直观理解开源软件（此处是指 R 和 Python 这类脚本语言为主的开源软件），如表 1.1 所示。

表 1.1 商业软件与开源软件的比较

	商业软件	开源软件
初学者难易	容易，记住点击方式	困难，需要熟悉脚本语言
熟练者难易	普通，需要学习新的模块	普通，需要修改代码或者内核
开发者难易	困难，封闭，不公开，黑盒子。	容易，开放，公开，供研究。
使用成本	付费使用	免费使用
安装	一键到底	需要调试
维护	公司负责	社群负责，基金会运营
知识产权	有，侵权究责	有，但开放共享绝大部分功能
改造可能	无	有，但需要社群认可
更新	更新相对稳定，速度相对平稳	更新相对不稳定，速度相对急切
占用计算资源	中等	低

其次，开源软件的获取，可以来自各个方面，因为任何网站都能传播该软件，因此人们一般选择进入官网下载。这些官网，在全世界都有镜像站，所以一般情况下，可以无障碍地访问网站和下载适合的软件版本。

- R Language：https://www.r-project.org
- Python：https://www.python.org

需要注意的是，市面上还有一些集成环境的套装软件，如 Anaconda、R Studio 等，是否选择开源软件，或者免费的集成套装软件，见仁见智，我们对比如表 1.2 所示。

表 1.2 套装软件与开源软件的比较

	集成软件	开源软件
初学者难易	容易，内嵌 Help 模块辅助	困难，需要熟悉脚本语言
熟练者难易	容易，内嵌 Help 模块辅助	普通，需要修改代码或者内核
开发者难易	困难，封闭，不公开，黑盒子	容易，开放，公开，供研究
使用成本	免费使用，特殊需求可付费定制	免费使用，一切自己搞懂搞定
安装	一键到底	需要调试
维护	无人负责，付费给公司会负责	社群负责，基金会运营
知识产权	有，侵权究责，但多数功能免费	有，但开放共享绝大部分功能
改造可能	无	有，但需要社群认可
更新	集成环境与软件版本绑定	更新相对不稳定，速度相对急切
占用计算资源	高	低

需要注意的是，集成套装软件，占用计算资源特别多，占用内存资源也是。其次是商业套装软件，最不占据计算资源的是开源软件，然而开源软件一旦加载过多的程序包，则会逐渐变得臃肿和占据更多计算资源。

另外，即便都是开源软件，也有不同。R Language 运行在操作系统上，所以计算机的内存，决定了 R 能够吞吐多少数据量以及具有多少计算能力；但是 Python 则是作为操作系统的一部分，因此，计算机的内存，就是它的缓存（数据吞吐量），而计算机的 CPU 等计算资源，则是可视为它的运算能力（在没有其他干扰的情况下）。对于开发者级别的用户而言，这些问题都已随着 SparkR、HadoopR 等的程序包的开发利用，以及 Python3X 架构重组，得到解决。

二、开源软件的使用

不像商业软件，已经规划好了使用范围、步骤和方式，在它的“世界”里，原则上不会出现什么因为软件操作所导致的问题；开源软件，让人们在使用过程中遇到各种各样的“小挫折”或者“小麻烦”的情况屡见不鲜。部分原因，如第三节所说，来自数据方面产生的困惑，部分原因，则是因为我们需要自行编写或者修改脚本语言的代码，所以偶尔会有调试和代码除错的任务。

解决一个个使用过程中出现的麻烦，是最快入门开源软件的方式和途径。所以，对于开源软件使用过程中需要解决的各种问题，不要抱持着拒绝除错或者拒绝分享的态度，而要采取欢迎问题和勇于解决问题的态度，才能实际有所收获。

因此，在开源软件的使用过程中，或者从初学者到进阶者的角色变化时，通常需要参与或者加入某些社区，根据自己的能力和水平程度，选择适合的社区加入即可，也可通过学校课程所组建的网络社区，找到相近水平，具有共同目标的同学们一起研究。既然我们有可能加入某些开源软件社区，那么就要注意“提问的智慧”和礼节，例如：

（一）能够用课件解答的问题，就先自查，或者帮助回答，不要动不动就在群里呼天喊地。

（二）同理，能在百度、知乎、CSDN 解决的问题，也可自查，或者借用引用来帮忙其他同学的问题。

（三）如果确定提问，务必把背景和前面做过的步奏描述清楚，比如，用什么数据（发上部分数据或者描述数据）、用什么程序包（例如时第几版的？配置环境是什么？）、用什么代码（发来完整版的代码，不要只截一句）。

这样，如果能够自己找到解决方案，就自己解决，因为开源软件的开发者社区，有一不成文的规定，即：不作同样的轮子，不犯同样的错误。许多 Bug 实际上已经有人处理或者找到解决方式了，在互联网上能够找到解答。

最后，如果加入社区，最好保持实名制，以及友好交流。往往人们几天没睡又得不到直接有效的解决，或者没想清楚因此表达问题不够准确，无论双方处于什么状态，都不要谩骂和发泄情绪，遵守法律法规的同时，也保持风度和气质。更何况，互联网是有记忆的。

三、开源软件的传播

在我们所接触的 R Language 和 Python 两种开源软件，他们虽然都是开源，并且两大开发者社区也在 2018 年开始合作，互相学习借鉴，不再重复建设。但是，两大社区的管理模式，也就是开源软件的版本发布以及附加程序包和库的模式，还是有些不同。我们对比如表 1.3 所示。

表 1.3 开源 R 和开源 Python 的社区比较

	R Language	Python

诞生的历史背景	统计软件价格日趋高涨	编程语言不够灵活好用
创始人背景	统计学家+计算机编程	黑客
社区管理	集中式，经过层层审核方式，较为学院派的方式进行管理。	集权，但放任式管理。 （现已成为少数人集体协商）
新的程序发放	提交 R 实验室，通过 R 组织的审核后，最终允许放置官网上。	自由发布，每个人根据自己所需以及其他人的总体评价予以选择。
成员贡献认可	官网，特别是 R -Journal 期刊	开源代码平台 GitHub 上的评分
社区成员组成	学者、学生、科学爱好者	极客、黑客、骇客（过去）
成员技术水平	科学水平较高，工程水平较低。	极端优秀者多，总体水平不平均

因此，两大社区各具特色，具有互补作用，因此当初我们考虑如何实践数据科学的时候，把 R 与 Python 一并纳入。早期，两大社区所开发的新的软件功能（程序包、库）并不同时，也不一致，所以在使用上可以互相弥补，后来，随着两大社区的逐渐相互学习与合作，我们可以从内核代码以及脚本语言的层面上，进行同一问题不同解决方案的比较。

此外，两大社区本身对于数据科学的贡献，分别采取了各自擅长的参与方式，这也是值得吸收和了解的。

第五节 数据科学入门途径

一、体系化的学习

我们把《数据科学 R 与 Python 实践》所涉及到的数据处理、数据分析、统计学、机器学习算法、大数据技术和人工智能等，原本分散在各类著作中的一组组概念和操作，视为一个完整的、相互关联的，根据第一章（尤其是第一节和第二节）能够贯穿起来的一个体系。

如果能够掌握这个体系，那么思考问题时，就不会仅从单一面向去设计解决问题的途径和方式，而且，在每个分支上，如果要再继续深化研究，也能够知道局部升华的实践经验，在整个体系中处于什么位置。

有鉴于此，我们根据图 1.4 的课时建议，设计如下表 1.2 的内容，提供教师或者自学者，熟悉掌握的捷径。

表 1.4 体系化学习《数据科学 R 与 Python 实践》的建议

章节	主题	学习方式
第一章	数据科学绪论	熟悉日后贯穿各章节的要点。
第二章	数据处理篇	手把手实操
第三章	数据型塑篇	手把手实操
第四章	数据可视化篇	部分实操，部分按照代码和数据完成。
第五章	统计分析篇	部分实操，知识记忆，动脑辩证思考。
第六章	数据建模篇	实操以及知识记忆
第七章	机器学习篇	实操以及知识记忆
第八章	大数据篇	给工具、讲解、讨论
第九章	人工智能篇	给工具、讲解、讨论

此外，由于九个章节的设计是层层递进的，因此，提供一副体系化学习的建议，如图。

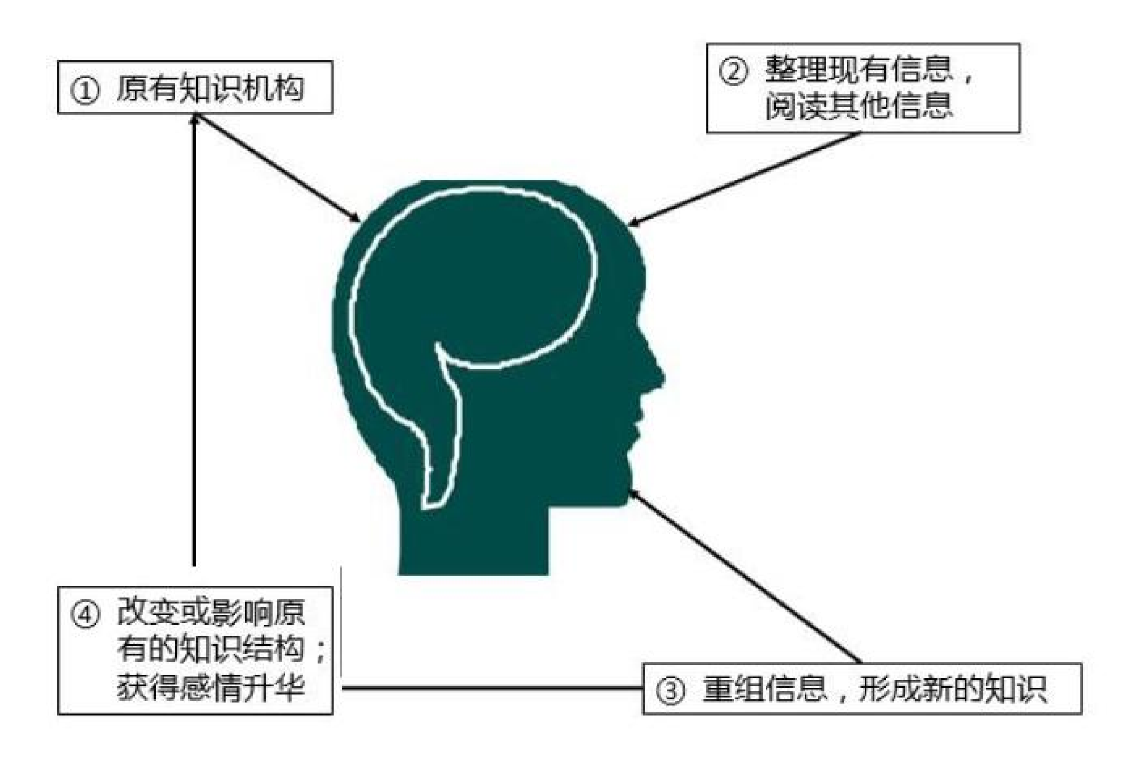


图 1.8 体系化学习的示意图

二、个体化实践

面向课题组，首先模仿前人研究，集中焦点在一、二位已经颇有成果的学者，甚至可能就是在学学生的指导老师等，关注他们使用哪些数据开展研究。

第二阶段，根据目的、方法、结论的三要素解析论文或者项目，其中的方法部分，着重分析采取什么方法论和研究方法，这类方法论和研究方法主要依靠什么类型的数据才能实现，因此，描述描绘这类数据的特征、格式和特点。

第三阶段面向数据，分析数据怎么来，数据怎么用，以及数据怎么管。

第四阶段，尝试转换其他数据，可能得自其他地方的开放数据，或者通过仪器观测、实验设计、观察采集等的数据集。把这些数据进行处理和可视化。

第五阶段，把数据集连同理论或者前人研究成果，重新复制做法，寻求使用数据科学的原理原则以及 R 和/或 Python 的实现手段，予以试验。

第六阶段，试验新的可能，进行探索实践，成为新的研究。

第七阶段，将上述过程进行经验总结，检查过程，提出改进要点。

第八阶段，与其他人进行交流分享，找到相同问题，找到各自优点。

第九阶段，在前面几个阶段的经验基础上，重新梳理第一章到第九章的内容，选择需要的内容，抛弃或者搁置没有用处或者影响自己的内容，提出要点，补充重点。

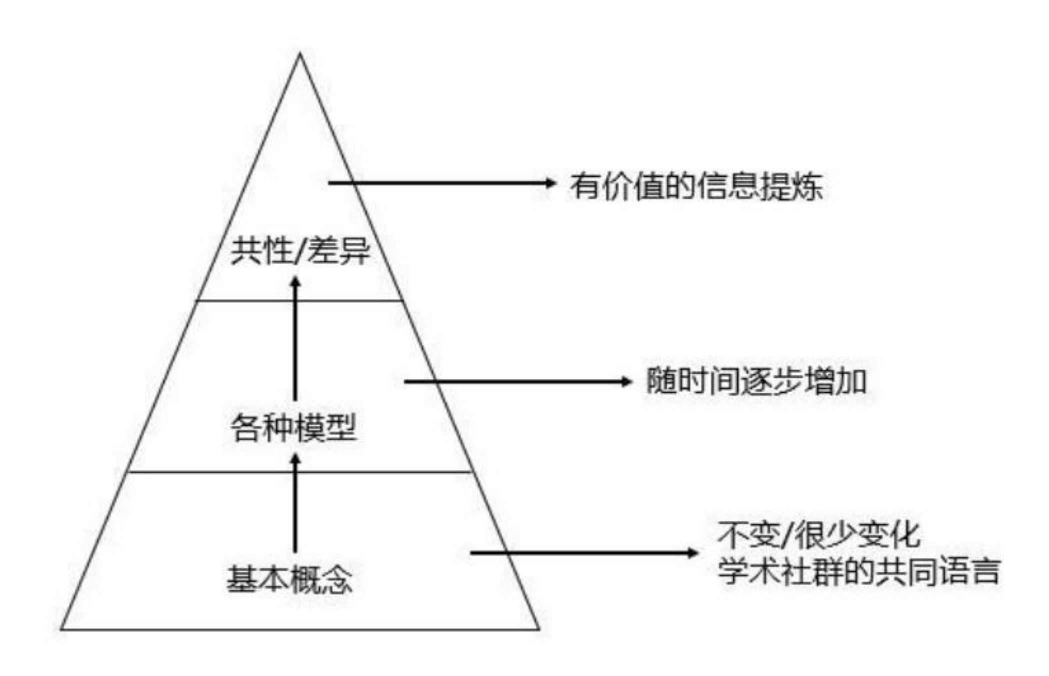


图 1.9 个体化实践的示意图

个体化实践搭配体系化学习，某人的个体化实践经验又与其他人的经验互为比较、参考和归纳，因此提供图 1.9 作为个体化实践的建议。

本章小结

本章介绍了数据科学的入门知识，对于之后各个章节的理解起到浓缩要旨的作用，主要讲述了以下内容：

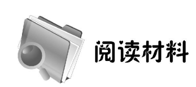
(1) 数据世界与真实世界，大致可以从三个方面考虑：哲学思维能够时时提醒我们任何数据科学的结论都是短暂而非永恒的；科学思维能够帮助我们理清结论如何而来，推演的限制范围、解释的力度和限度等；工程思维能够直接帮助我们设计如何得出结论。

(2) 数据科学的思路、方法与流程，旨在强调数据驱动科研，通过辩证性思维去逼问最接近的答案并且予以多方多次多番检验。

(3) 开放数据的获取、利用和传播，重点在查清楚数据集的来历以及给于正确引用，并且如果是自产数据，则可以通过一系列注册方式，予以开放共享，给更多人使用和引用。

(4) 开源软件的获取、利用和传播，重点在比较之后选择适合自己的开源软件，在使用过程中注意参加社区讨论的礼节，以及认识到不同社区合作为数据科学作出贡献的可贵。

(5) 数据科学的入门途径，大致可从两个方面共同着手，一是有系统地按部就班进行学习和复习，二是面向课题进行模仿、转换、试验和创新的独立探索实践。

 **阅读材料**

数据作假

（1）有些人，把实验和观察中不支持自己观点的数据删除，以得到有利于自己观点的结论，诸如“结果显著”之类的结果，这种行为明显违背了科学研究的基本原则。

（2）有些人，发现数据存在若干“异常值”时，就以数据清洗为名，进行删除，而用剩下来的“干净的”数据来拟合想象中的模型。这种做法并不严谨。
（3）通常，最好先了解这些“异常值”的来源；有可能“异常现象”反而是很有价值的信息；如果是实验或者观察过程中的噪音污染，对于模型或者假说或许还需要增加更长的研究时间，但对行业发展的经验则是有利。

习　　题

一、填空题

1、科学研究的核心在于___变量___之间的关系，把能够___观测___的事物予以___计算___，形成简化后的模型，来解释甚至预测世界。

2. 数据，是关于___变量___的观测值。例如，有人通过社会科学调查方法，调查产生一些数字，这些数字就是数据（data）。___变量___只有用数量来描述时，才有可能建立数学模型，并使用计算机来分析。　。

3、模型的目的是：___解释___、___预测___，或者理解产生数据的机制。。

二、 多选题

1.关于数据驱动、问题驱动、用户驱动的形象化方式，正确顺序是（___）
（1）针对模型产生的新的信息进行更新、判断、解读。
（2）根据数据建立模型；
（3）面对问题收集数据；
（4）利用模型做预测或者得到结论；

2.观察数据集的变量关系，正确顺序是（___）
（1）两个变量的关系是否显著？
（2）两个变量是否有关系？
（3）这个关系是不是因果关系
（4）这个关系是否带有普遍性？

三．问答题

1、使用他人的开放数据应该注意什么？
2、开源软件的使用方式包括哪些方面？

四．课后作业

请下载您们各自的指导老师的论文，要以他为主的论文，共 10 篇论文。
压缩 PDF 文件为一个.zip 文件，发到导师邮箱。
邮件标题：您的姓名和学号_第一次作业。例如，李三脚 200619780816001_第一次作业

1.作业释疑：什么叫做【以他为主的论文】呢？
各个学科认定不同，一般而言，大致这么判断：
（1）导师作为独立作者的论文
（2）导师作为合作作者，并且是第一作者的论文。

（3）导师作为合作作者，并且是第二、第三作者的论文。
（4）导师作为合作作者，该论文的合作作者均为导师课题组成员
（5）导师作为通讯作者。

2.作业释疑：如果老师发表超过 10 篇论文了，怎么筛选呢？
这种情况应该是多数，一般而言，大致遵循从（1）到（5）的筛选方式：
（1）导师曾经推荐给您的论文，他是该论文的作者之一。
（2）导师的论文（参考问题 1 的解答 1 的认定方式）中，有用到【数据】的论文。
（3）导师发表在较高水平期刊上的论文。
（4）导师曾经发表的论文，而且被引数比较高的那几篇，越高越好。
（5）导师最近发表的论文，按照时间远近，越近发表的越好。
*其中，（4）和（5）要综合考虑。

3.作业释疑：什么是“有用到【数据】的论文”呢？
（1）这意味着，您的导师所独立研究撰写、所领导指导小组的研究，或者所参与的研究，有利用到“科学数据”进行实证研究后，所撰写的论文。
（2）除非导师曾经指导您看他的综述性论文，否则一般我们想要分析的是，他以及课题组怎么利用数据进行科学研究，而您上这门课程之后，可以怎么应用。
（3）如果导师发表过“数据论文”的话，那篇数据论文，可以优先作为 10 篇论文之一。
（4）导师或者您所在的课题组，可能采用自己课题组的数据，或者其它课题组的数据，这都不影响选不选择作为 10 篇之一，我们需要确定的是，是否使用了数据进行研究。
（5）导师或者您所在的课题组，可能使用原始数据、衍生数据，或者研究数据，进行研究后，发表若干论文，无论使用哪种数据，都不影响选不选择作为 10 篇之一。

第二章　数据处理篇

学习目标

掌握如何创建数据表格；
掌握如何存取数据表格；
掌握如何读写不同格式的数据集；
掌握如何转换不同格式的数据集；
掌握如何撰写和保存脚本语言实现自动化；
掌握如何读取和修改脚本语言实现自动化。

知识结构

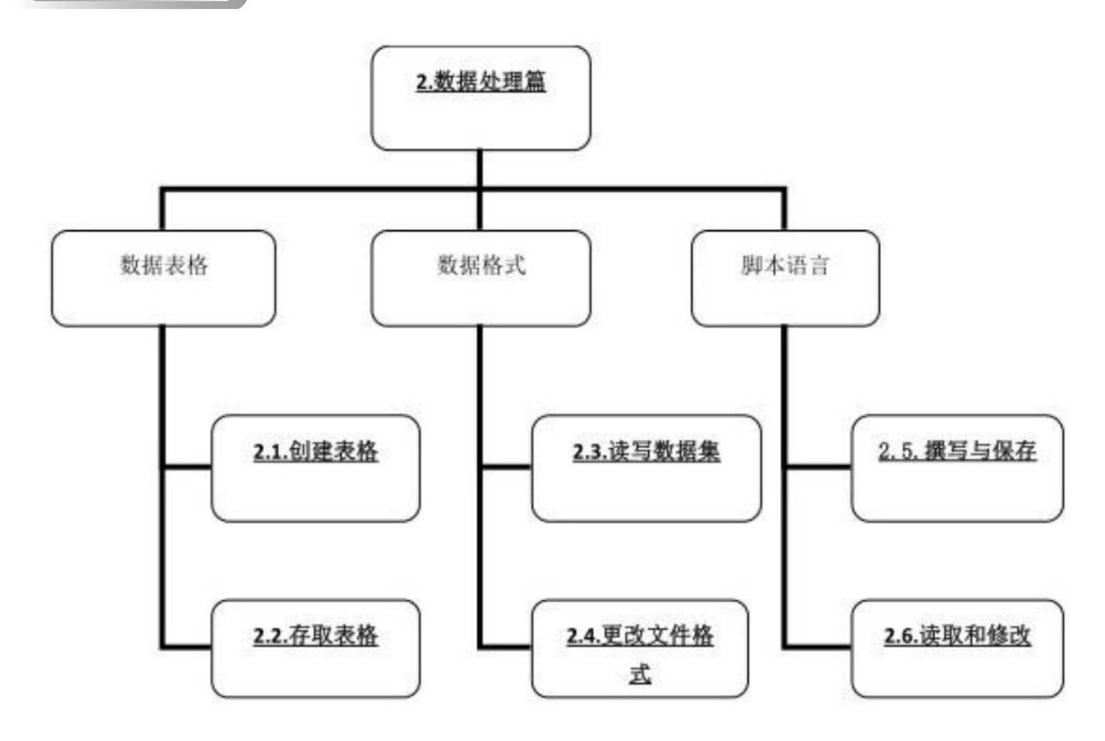


图 2.1 数据处理篇的概述图



导入案例

当我们开始处理数据之前，我们应该是会得到一笔数据，或者一组数据（又称为：数据集），它们具有各种各样的格式，比如：文字形式、数字形式、图片，或者音频视频等，它们的数据量大小，也有很多种情况。

人人几乎手上都有一些数据，人人都可以就自己拥有的数据进行一些处理，换言之，每个人都有自己独特的数据管理的方式；但是，如果我们要在数据科学的背景下，去讨论一些共通原则，或者比较便利、比较有效率、比较能够相互进行数据共享等的情况，那么，我们就需要引入一些工作流程作为彼此之间进

行数据科学之前的共同基础。

如同第一章第四节的介绍，我们可以从 R 的官方网站，和 Python 的官方网站，下载并且安装 R 与 Python 的软件，通过这两个软件的操作，来理解数据处理的工作流程以及基本概念。

R Language：https://www.r-project.org

Python：https://www.python.org

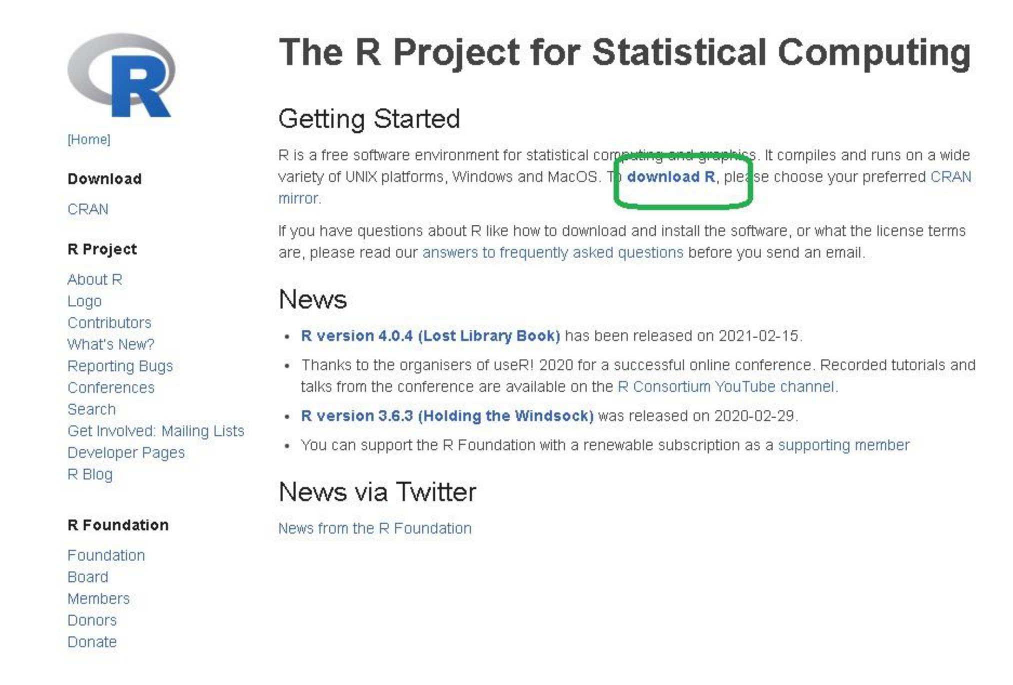


图 2.2 前往 R 的官方网站下载

进入官网之后，我们直接点击 download R 即可，进入镜像站后，选择任意其中一个网址，进入之后，按照您的计算机的操作系统的版本，选择合适的 R 进行下载安装。

R 语言是非常丰富的一门语言，在初学者阶段，我们只需要跟随安装过程中的提示，一步步安装 R 软件，能够启动操作，即可。

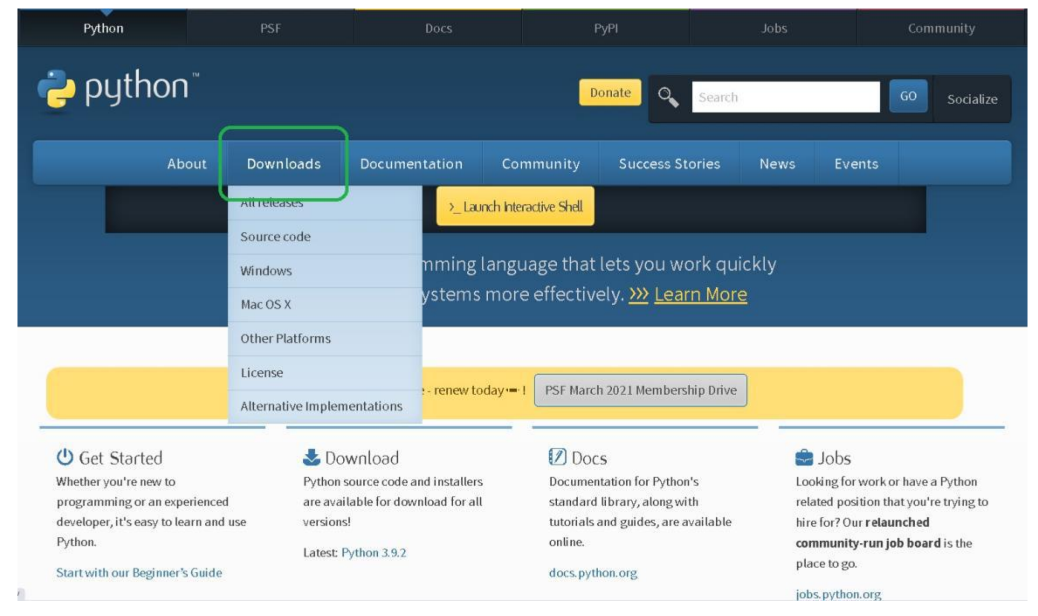


图 2.3 前往 Python 的官方网站下载

同理，我们进入 Python 官网之后，直接点击 downloads 并且选择合适的版本，进行安装即可。在 Python 3.7 之后的版本，已经能够直接安装路径。早前 Python2.7 版本以及 Python3.7 版本，需要自己进入操作系

统的 PATH 进行设置。

在安装 R 和 Python 之后，我们可以开始通过数据表格的创建、存储以及读取，来进行数据处理的基本流程的操作。这样，我们可以通过实际操作，体验什么是数据处理。

然而，往后所接触的数据集的格式，不一定每次都是数据表格（数据框 Data Frame）的形式，也有其他各种可能，所以，在我们熟悉了在 R 和 Python 也处理一般商用统计软件的数据表格之后，我们就需要知道如何进行数据格式的转换。

因此，本章第一节到第四节主要学习数据表格的一般操作，在第五节和第六节则是进一步熟悉 R 和 Python 的一般操作。为之后的章节打下基础。

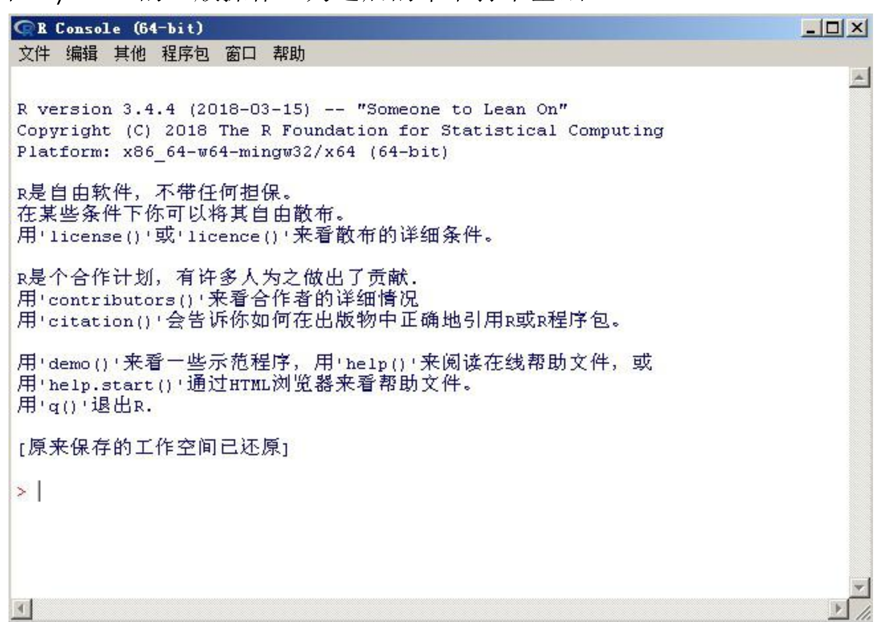


图 2.4 安装好 R 的情况

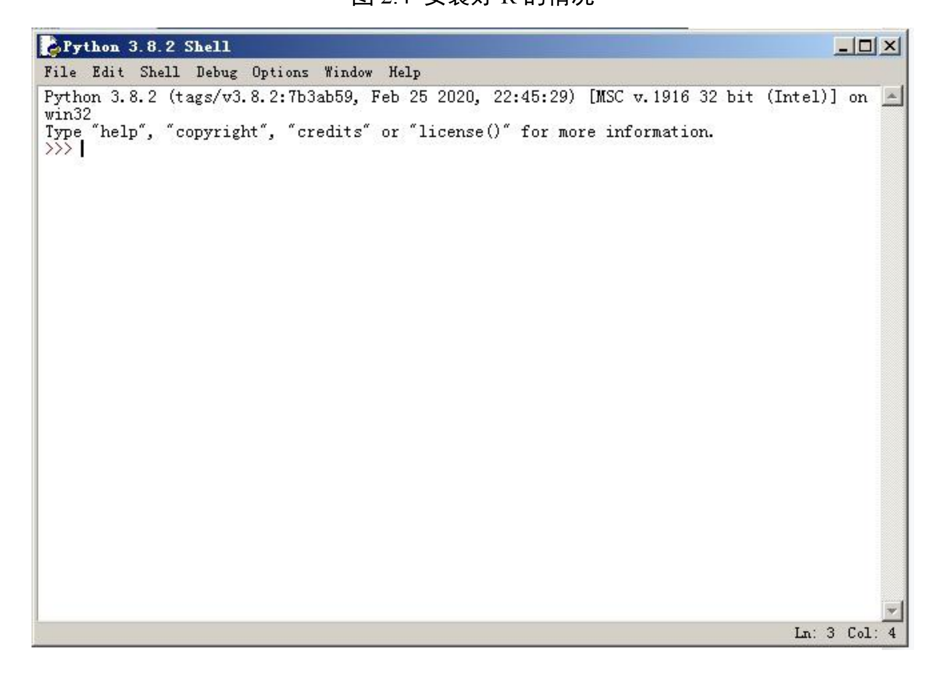


图 2.5 安装好 Python 的情况

第一节 创建数据表格

一、创建数据

创建数据几乎是所有讨论数据分析、数据挖掘或者数据管理的第一个步骤或者第一件事；然而，我们在《数据科学 R 与 Python 实践》的课程里，并不太强调它的重要性；与之相反，我们只是通过创建数据的过程，来认识我们用来解释和创造数据世界的两种语言：R 与 Python 而已。我们不把 R 和 Python 视为一种工具，或者是数据工程的利器，而是把它们视为两种具有沟通用途的语言。

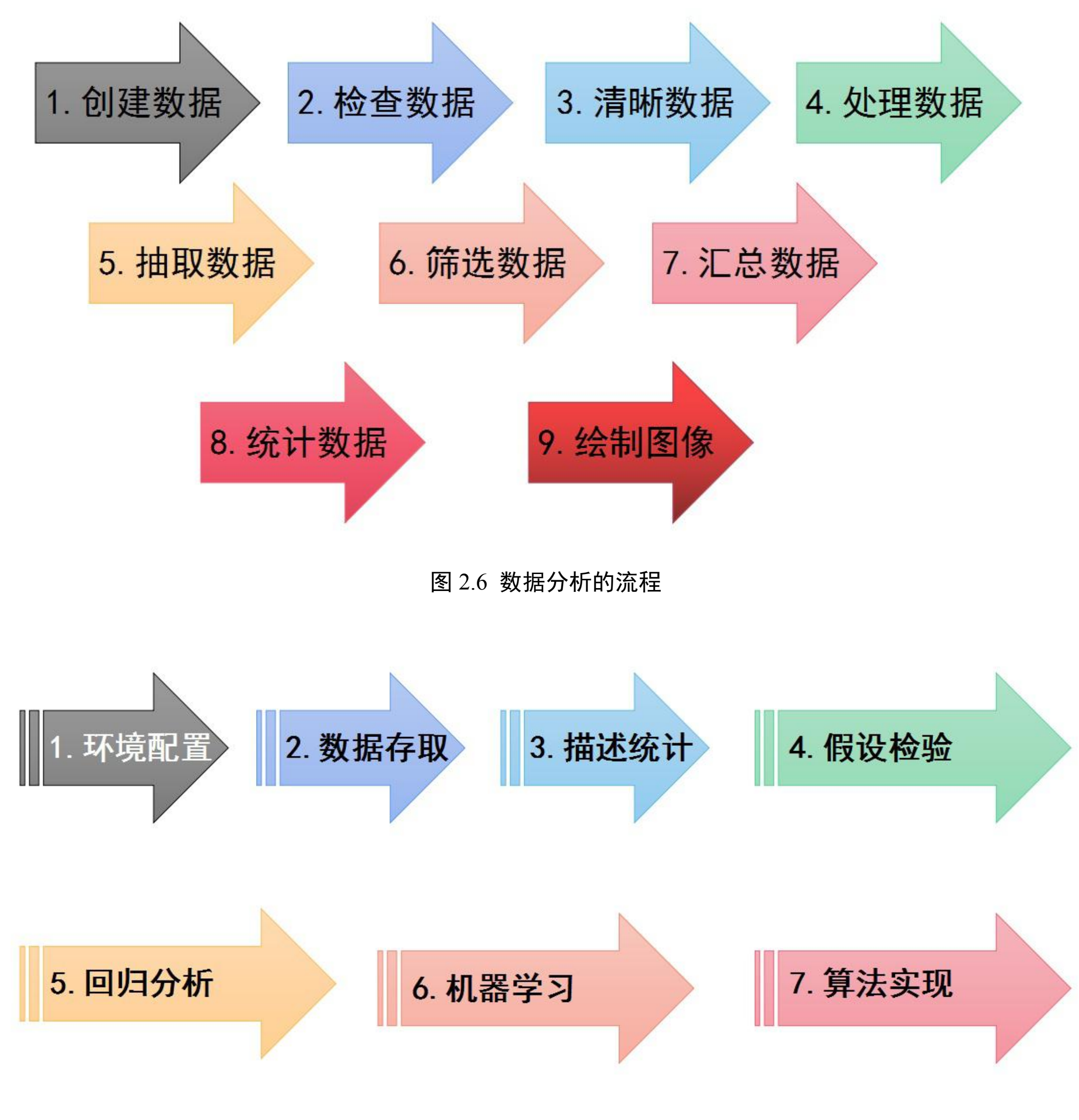


图 2.6 数据分析的流程

图 2.7 数据挖掘的流程

当我们开始处理数据之前，我们应该是会得到一笔数据，或者一组数据（又称为：数据集），它们具有各种各样的格式，比如：文字形式、数字形式、图片，或者音频视频等，它

们的数据量大小，也有很多种情况。最重要的是，它们具有不同的数据类型，每种语言都有它擅长或者优先研发出来的数据处理的方式。

当我们接触 R 语言，经常首先学习：数据框（dataframe）、列表（list）、矩阵（Matrix）、数组（array）等，便于进行统计计算。

当我们接触 Python 语言，经常首先学习：字符串（str）、元组（tuple）、列表（list）、集合（set）、字典（dict）等数据类型。

这里我们暂且放下各种数据类型、各种数据处理的计算机技术，以及各种编程语言的差异的理解；为了便于入门，我们先以数据表格作为数据处理的学习起点，操作并且比较 R 和 Python 的异同。



阅读材料

面向对象编程

【面向过程（procedure-oriented）】：以事件为中心的编程思想，把解决问题的步骤写出来，程序一步一步执行就能解决问题。

【面向对象（Object-Oriented）】：以事物为中心的编程思想，把问题相关的数据提取出来，将具有相同属性的物体抽象为【类】，并给“类”设计相应的方法。

面向对象的编程，在程序执行时，通过创建某个类的一个对象，调用这个类的方法，进行解决问题的实现。

R 与 Python 都是面向对象编程，所以，接下来，我们在代码里的 x1,x2...f 等，都是【对象】。

二、用 R 创建数据表格

我们首先采用【赋值】的方式，即，用<-把右边的数值，赋予到左边的对象。在下列代码中，对象是 x1，x2，x3，f 而<-右边则是要被赋予这些对象的数值。

这些数值，包括字符串，如：Alan、Irina、M、F 等，需要加上引号‘’；也包括数字数值，如：1000、2000 等。

把这些信息予以整理起来，制作成为表格形式，在 R 语言里，惯用的是【数据框】（Data Frame）函数。

总之，我们想要整理一些信息的话，可以参考如下代码：

```
x1 <- c("Alan","Irina","Mark","Lisa")
x2 <- c("M","F","M","F")
x3 <- c(1000,2000,500,1500)
  # 建立了 x1, x2, x3 三个对象。
  # 函数 c( ) 是内嵌在 R 的一个函数，具有合并的意思和功能。
f <-data.frame(Person=x1, Sex = x2, RMB = x3)
f
  # 函数 data.frame()是内嵌在 R 的另一个函数。
  # 最后一行 f 是为了打印出来，看看我们整理得如何。
```

如果只有 x1 等的对象，作为一列列的变量，还不足够，因此我们需要用到函数 data.frame 来进行【二维数组】的组织。

在函数 data.frame 当中，我们将 Person, Sex 等的名称【命名】x1...x3 等对象以及

它们所代表的数值。

注意：命名（names）和赋值（assign）在很多地方，看起来功能相同，但是实际上并不一样；在 P 和 Python 里，表达含义也略有不同，我们会在第四章讲解。

此处，我们能够键入代码，完成一组数据框，就可以了。目前不必过于强调细节，如果按照上面五行代码，应该出现：



图 2.8 用 R 创建数据框

以上，我们已经用 R 创建了一个数据框，从人类视角来看，是做了一番信息整理之后的数据表格。

三、用 Python 创建数据表格

在 R 里能够做的事情，在 Python 里也能做到。以下代码，就是制作与前面第二部分相同的数据表格。

```
import numpy as np
import pandas as pd
from pandas import DataFrame
# 导入 numpy 库以及 pandas 库。
## 导入 pandas 库之后，从中在提取 DataFrame 函数，便于更快使用该函数。

data = {'Person': ['Alan','Irina','Mark','Lisa'],
    'Sex': ['M','F','M','F'],
    'RMB': [1000,2000,500,1500]}
# 此处，以 Python 数据类型的【字典】形式，进行【数据框】的制作。
# 字典，即符号 : 左右两边，一边是【键】（key），一边是【值】（value）。

f = DataFrame(data)
f
```

借由上述代码，我们解释两个知识点：

（1）因为函数 data.frame 内嵌在 R 里，因此在前面第二部分的操作中，我们直接使用了这个函数；但是，在 Python 并没有内嵌数据框的函数，因此，我们需要加载 pandas 库之后，在 python 导入（import）库 pandas 之后，才能使用这个函数及其功能。

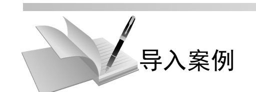
（2）其实，在 python 里，所内嵌的数据类型中，字典（dictionary）和序列（Series）都可以用来进行 2.1.2 的信息整理，但是多数用户习惯了 SQL 和 R 的这种数据框的方式，因此我们不妨直接利用 pandas 进行相同类似的处理。

（3）符号 # 在 R 与 Python 的用途相同，都是用来“注释代码”的，所以计算机并不会执行 # 之后的信息，会自动忽略掉 # 之后的代码。

如果我们实现上述代码，应该出现和第二部分一样的数据表格，如下：

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import DataFrame
>>> data = {'Person': ['Alan','Irina','Mark','Lisa'],
        'Sex': ['M','F','M','F'],
        'RMB': [1000,2000,500,1500]}
>>> f = DataFrame(data)
>>> f
  Person Sex   RMB
0   Alan   M  1000
1  Irina   F  2000
2   Mark   M   500
3   Lisa   F  1500
>>> 
                                                              Ln: 16  Col: 4
```

图 2.9 用 Python 创建数据框



导入案例

此处，我们制作一副数据框（Data Frame），用来记录 Alan, Irina, Mark 和 Lisa 四人的性别（Sex）以及收入（RMB）。据此，一步步开展日后的工作。

如果数据量很大，一般来说，采用的是本章稍后用到的数据处理技巧，而不是一个个数值键入。

事实上，在 R 里也有内嵌一组函数，可以让我们在视窗环境下，进行信息输入以及数据表格的创建。当初为了 Excel 和 SPSS 等用户，能够适应 R 语言的环境，所建立，现在 R 的使用者里，很少有人使用。

第二节 存取数据表格

一、数据的存储与取用

当我们完成创建数据的基本工作后，就可以进行数据存储了，这些操作与一般商业统计软件或者办公软件等，并无二致。特别是试算表（Sheet）可以看作是数据表格，或者数据框（data frame）的另一种形式和名称。

在数据科学领域、数据管理（Data management）领域、数据监管（data curation）领域等，有一个重要概念，就是：数据版本（data version）。最简单直接的理解方式，就是每次进行数据存储，均有不同的文件名称，就有不同的数据版本。当然，数据版本还包括了：数据类型、文件格式、数据模型等一系列的内容，我们都在本章予以介绍。

在《数据科学 R 与 Python 实践》的课程里，我们对数据版本并不过多着墨，因为认为这是自然而然，在数据处理的过程就会注意到的事情。当然，如果没有意识到这件事情，那么还是需要仔细考虑：在每次处理数据之后，是否有必要存储不同的版本，以备追溯每个版本的变化过程。（参考 1.最末：《数据造假》的内容）

二、用 R 存取数据表格

我们把第一节第二部分所做的数据框，应用函数 write 进行【写入】的任务。因为写入的文件格式不同，所以在 write 后面，会接上.csv 代表以 csv 格式存储。代码如下：

```
write.csv(f,"C:/Users/Hp/Desktop/f.csv",row.names=FALSE)
# 写入.csv 文件。
## 把 f 这个对象，写到 C:/...路径下的 f.csv 文件。
## row.names = FALSE 是指：名称的列，不要写入。

f1 = read.csv("C:/Users/Hp/Desktop/f.csv")
f1
# 读取 f.csv 这个文件。
## 将读取 f.csv 文件的内容，赋予到 f1 这个对象。
```

在 R 和 Python 里，存储和取用文件，都要标明【路径】（direction），也就是文件的名称以及所在的位置。如果我们设定好了工作路径（work direction）并且经常只在工作路径里，存储和取用相关文件，则不必每次都写上路径。因人而异。

真值（TRUE）和假值（FALSE）是一组逻辑，此处不赘言描述，在函数里，如果没有特别标示 FALSE 则一般默认为 TRUE 的。例如，上面代码，row.names = FALSE 注明：名称的列，不要写入；同时，默认了行的真值。

完成上述代码后，应该出现如下情况：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> write.csv(f,"C:/Users/Hp/Desktop/f.csv",row.names=FALSE)
>
>
> f1 = read.csv("C:/Users/Hp/Desktop/f.csv")
> f1
  Person Sex  RMB
1   Alan   M 1000
2  Irina   F 2000
3   Mark   M  500
4   Lisa   F 1500
>
```

图 2.10 用 R 存取数据框

这样，我们通过 f1 验证了 f.csv 的存在，并且能够对照 f 和 f1 来观察出，在刚刚的数据处理中，哪怕只是存储和取用两个步骤，也没有改动数据本身。这是很重要的一件事情。接着，我们就可以用 Python 进行相同任务。

三、用 Python 存取数据表格

因为 Python 设计之初，并非为了办公软件、试算表软体、统计分析等功能，更多目的和关注点在计算机编程语言，特别是脚本语言，为了让更多人便于计算机编程。因此，我们在用 Python 时，思路偶尔与 R 不同（习惯了，差别也不大）。

为了存储数据框，我们需要先设定路径，然后利用 pandas（之前已经安装，并且在第一节第三部分导入的库）的函数.to_csv()把对象 f 写入.csv 文件。

其后，应用函数 pd.read_csv()读取.csv 文件。

代码以及代码注释，如下。

```
outputpath='C:/Users/Hp/Desktop/f.csv'
# 设定路径

f.to_csv(outputpath,sep=',',index=False, header=True)
# 将对象 f 存储为 csv 文件。
## 以逗号 , 区隔 f 对象里的数值，索引 index 不要，名称的列需要要。

f1 = pd.read_csv('C:/Users/Hp/Desktop/f.csv')
f1
# 读取 f.csv 这个文件。
## 将读取 f.csv 文件的内容，赋予到 f1 这个对象。
```

注意，比较一下 R 和 Python 在读取数据框的函数的不同表达，可以便于理解两种语言的不同风格和设计。

在 R 里，我们通过 write.csv()和 read.csv()进行读写。在 Python 里，我们通过.to_csv()和 pd.read_csv()进行读写。有什么不同呢？

在 R 里，内嵌了一系列数据处理的函数，如 write.csv()和 read.csv()在形式和用法上，都一致一样。而在 Python 里，我们先是 f.to_csv() 作为一个模块，进行写的过程（还

记得在此之前要有一个设定路径的 outputpath 么？），然后，再在 pd.read_csv()上，通过缩写的 pd 调用 pandas 库，在库里面通过函数.read_csv()完成步骤。

当然，我们可以改写代码，使得 Python 的这段代码更像 R 的简练，也可以改写 R 的那段代码，使得它更像 Python 这段代码那样，清晰可见文件系统架构的脉络。不过，为了同时保持计算机和统计学的敏感度，我们不做这类改写。

完成上述代码后，应该出现：

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>>
>>>
>>> outputpath='C:/Users/Hp/Desktop/f.csv'
>>> f.to_csv(outputpath,sep=',',index=False, header=True)
>>>
>>>
>>> f1 = pd.read_csv('C:/Users/Hp/Desktop/f.csv')
>>> f1
  Person Sex   RMB
0   Alan   M  1000
1  Irina   F  2000
2   Mark   M   500
3   Lisa   F  1500
>>> |
                                        Ln: 29  Col: 4
```

图 2.11 用 Python 存取数据框

果然，图 2.11 和 2.10 几乎外观上没有差别，我们存储的 f.csv 文件里的数据集，也没有不同。此时，我们可以说，两种语言，至少在数据处理的数据存取上，是可以相通的！


阅读材料

编程语言要解决什么问题？

一言以蔽之，用机器解决复杂行为。

把人类智慧写出来，写成 Python 的“库(library)”之后，或者写成 R 的“程序包(Package)”之后，分享他人并且累积起来。

这样，人们就不必重复同样的智力劳动，而只需要关注已有的 library 和 package 怎么应用，以及如果不能解决问题，应该开发什么新的模块。

这就是集体智慧（collective intelligence）的力量。所以，我们不把 R 和 Python 当作是数据科学的工具，而是把它们当作描述、解释和发展数据科学的语言。

第三节 读写数据集

一、数据读写

在第一节和第二节中，我们通过 R 和 Python 分别创建了相同的 f.csv 文件，这里有三层含义：

（1）f 是一个对象，被写成 f.csv 文件。这个.csv 是一个数据格式、文件格式。

（2）f 这个对象，是一个 Data Frame（在 R 语言里，用 data.frame 这个函数进行数据处理，在 Python 里，因为没有内嵌处理数据框的函数，所以要用 Pandas 库，在 python 的 Pandas 里，用 DataFrame 这个函数处理）。请注意，数据框，在英文、R 和 Python 三种语言里的不同写法。

（3）f 这个数据框，实际上表示了我们记录的 Alan, Irina 等四个人的信息，在日常生活中，我们通常叫它：数据表格。如果您记得第一章第一节的内容，请注意，数据框是在数据世界里，数据表格是在真实世界里，尽管所指的是同一件事。

在第三章里，我们会更改 f.csv 的 f 部分，也就是把 f 这个对象、数据框、数据表格，进行修改，因此，第三章称为：数据型塑。

然而，我们现在要做的数据处理，是 f.csv 的.csv 部分。换言之，如果我们能够利用 R 和 Python 读取各种数据格式的文件，又能够写成各种数据格式的文件，我们就能够把 f 变成各种不同的数据格式的文件。

二、用 R 读写数据集

与第二节类似，我们把数据框，不以数据表格的形式进行存储，而以文字文本的形式予以存储。使用的是 write.table()函数。代码如下：

```
write.table(f1,"C:/Users/Hp/Desktop/f2.txt",sep='\t',row.names=FALSE)
f2 <- read.table("C:/Users/Hp/Desktop/f2.txt",sep = '\t',header = TRUE)
f2
```

此处，我们在代码里面，增加了 sep 的【参数】，是为了把数值进行分隔，否则每一行的不同的列，将被视为同一个列。在正则表达式，符号\t 代表了分隔符，所以我们采取了 sep='\t'的作法。

另，在读取.txt 的文件时，我们把默认 header 为 FALSE 的情况，修改为 TRUE 了，这与前面第二节所说的，通常默认为 TRUE 的情况不同。

如果完成代码，应该出现：

```
R Console (64-bit)
文件  编辑  其他  程序包  窗口  帮助

>
> write.table(f1,"C:/Users/Hp/Desktop/f2.txt",sep='\t',row.names=FALSE)
> f2 <- read.table("C:/Users/Hp/Desktop/f2.txt",sep = '\t',header = TRUE)
> f2
  Person Sex  RMB
1   Alan   M 1000
2  Irina   F 2000
3   Mark   M  500
4   Lisa   F 1500
>
>
> |
```

图 2.12 用 R 读写数据集

如图 2.12 所见，我们能够用 R 读写.csv 和.txt 了，表示两种文件格式之间的转换，不成问题。



阅读材料

用 R 与各种统计软件沟通

作为开源软件的 R 语言，原本就是为统计工作，进行设计和开发。统计作为数据科学的核心之一，因此 R 语言很适合作为数据科学的一门语言。

统计软件	我们用 R 读取它们的数据格式文件的函数
SPSS	read.spss
SAS	read.ssd
Minitab	read.mtp
Stata	read.dta
Systat	read.systat
…	…

三、用 Python 读写数据表格

同理，我们按照第二节的 Python 存取.csv 所用到的思路，如何应用 pandas 的函数和方法，以及前面部分的 R 读写.txt 文件的思路，注意分隔符的参数 sep='\t'的方法。把代码撰写和运行，如下：

```
f1.to_csv('C:/Users/Hp/Desktop/f2.txt',sep='\t',index=False)
f2 = pd.read_csv('C:/Users/Hp/Desktop/f2.txt')
f2

f3 = pd.read_csv('C:/Users/Hp/Desktop/f2.txt', sep='\t')
f3
```

应该出现：

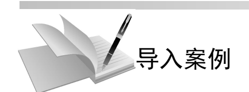
```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>>
>>> f1.to_csv('C:/Users/Hp/Desktop/f2.txt',sep='\t',index=False)
>>> f2 = pd.read_csv('C:/Users/Hp/Desktop/f2.txt')
>>> f2
  Person\tSex\tRMB
0     Alan\tM\t1000
1    Irina\tF\t2000
2      Mark\tM\t500
3     Lisa\tF\t1500
>>> f3 = pd.read_csv('C:/Users/Hp/Desktop/f2.txt', sep='\t')
>>> f3
  Person Sex   RMB
0   Alan   M  1000
1  Irina   F  2000
2   Mark   M   500
3   Lisa   F  1500
>>>
                                                    Ln: 36  Col: 0
```

图 2.13 用 Python 读写数据集

比较图 2.10、2.11、2.12、2.13 的话，我们会知道，在 Python 和 R 语言之间，在.csv 和.txt 文件格式之间，不存在数据处理的障碍。

那么，为什么不只使用一种语言、一种格式呢？

因为，从事数据科学的实践经验告诉我们，通常我们的合作对象，很难和我们采用相同语言、相同文件格式，甚至相同的代码和思路，所以我们需要在第二、三、四章，掌握一些基本的共同技能和共通能力。



导入案例

到底需要了解多少编程语言？

熟悉一种，远比很多种，每种都是浅尝即止，要好。

如果一定要有所取舍，那么：

（1）作为工作语言，多多利用 Python 处理日常业务。

（2）作为科学语言，多多关注 R 的各种最新进展。

（3）想要了解计算机原理，就需要了解 C 语言，不必成为专家，但要了解。

（4）如果可以，在 C 语言之后学习 lisp 语言，了解思想。

（5）尽管：lisp 在生产线运用的少，但能深入理解数据与数据分析；而且，

（6）Python 和 lisp 在程序语言表达，几乎相同 :-)

第四节 更改文件格式

一、 数据文件的更改

尽管本章第一节能够创建数据框，而第二节和第三节已经能够把数据框存储、取用、写入和读取在.csv 和.txt 以及以此类推到其他更多的文件格式上，但是如果想要更为灵巧地进行数据处理，那么就需要知晓列表（list）的基本用法。

列表（list）是编程语言里，最核心不可或缺的数据类型。简言之，列表能够变成其他数据类型，而且“装得下”其他任何数据类型的数据，可是，反之不一定必然。

如果考虑第一章第一节的内容，我们可以想像一下：数据框具有行列，就像书柜一样，里面可以存放很多大大小小不同的书籍，只是我们希望它们能够整齐一点；而列表则是一间拥有无限大的空间的图书馆，其中的每一层，既可以摆放一本书，也可以摆放一排书柜，也可以是另一间图书馆阅览室。

所以，列表是可以收纳各种数据类型数据集的对象。前提是我们要能放进去，也要记得位置，才能找出来；此外，计算机的实际内存和计算能力，也要跟得上的话，几乎没有列表不能处理的数据。

回到数据处理的话题，我们接下来通过 R 和 Python 的各自的列表，把两组在第二节和第三节所做的数据框，放在一起，然后在把来自.csv 的数据框，写到.txt 里，把来自.txt 的数据狂，写到.csv 里。这样，我们就能确保，我们能在一个列表里，进行第二节到第三节的所有可能的数据处理了。

二、 用 R 更改文件格式

如前所述，我们处理的 f1 和 f2 要放到一个 f3 的列表里，代码如下。

```
f3 <- list(f1,f2)
f3
```

完成后，应该出现：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> f3 <- list(f1,f2)
> f3
[[1]]
  Person Sex  RMB
1   Alan   M 1000
2  Irina   F 2000
3   Mark   M  500
4   Lisa   F 1500

[[2]]
  Person Sex  RMB
1   Alan   M 1000
2  Irina   F 2000
3   Mark   M  500
4   Lisa   F 1500
```

图 2.14 用 R 把两个数据框装入一个列表

接着，我们可以利用 R 的数据文件.rdata 进行中转处理。

文件.rdata 的特性是较少存储空间，并且在 R 里面的处理速度较快。

读写的代码如下。

```
save(f3,file = "C:/Users/Hp/Desktop/f3.rdata")
load(file = "C:/Users/Hp/Desktop/f3.rdata")
f3
```

完成后，应该出现：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> save(f3,file = "C:/Users/Hp/Desktop/f3.rdata")
> load(file = "C:/Users/Hp/Desktop/f3.rdata")
> f3
[[1]]
  Person Sex  RMB
1   Alan   M 1000
2  Irina   F 2000
3   Mark   M  500
4   Lisa   F 1500

[[2]]
  Person Sex  RMB
1   Alan   M 1000
2  Irina   F 2000
3   Mark   M  500
4   Lisa   F 1500
```

图 2.15 用 R 把列表变成.rdata 文件

接着，我们进行类似第 2 节的代码如下。

```
write.table(f3[1],"C:/Users/Hp/Desktop/f4.txt",sep='\t',row.names=FALSE)
```

```
f4 <- read.table("C:/Users/Hp/Desktop/f4.txt",sep = '\t',header = TRUE)
f4

write.csv(f3[2],"C:/Users/Hp/Desktop/f5.csv",row.names=FALSE)
f5 = read.csv("C:/Users/Hp/Desktop/f5.csv")
f5
```

应该出现：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> write.table(f3[1],"C:/Users/Hp/Desktop/f4.txt",sep='\t',row.names=FALSE)
> f4 <- read.table("C:/Users/Hp/Desktop/f4.txt",sep = '\t',header = TRUE)
> f4
  Person Sex  RMB
1   Alan   M 1000
2  Irina   F 2000
3   Mark   M  500
4   Lisa   F 1500
> write.csv(f3[2],"C:/Users/Hp/Desktop/f5.csv",row.names=FALSE)
> f5 = read.csv("C:/Users/Hp/Desktop/f5.csv")
> f5
  Person Sex  RMB
1   Alan   M 1000
2  Irina   F 2000
3   Mark   M  500
4   Lisa   F 1500
```

图 2.16 用 R 从列表中拿出数据形成.csv 和.txt 文件

三、用 Python 更改文件格式

代码如下。

```
f2 = [f1]
f2
f2.insert(1,f3)
# 因为 0 的位置（第一位）是 f1 所以 f3 我们放在 1 的位置（第二位）
f2
```

应该出现：

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>>
>>> f2 = [f1]
>>> f2
[    Alan  M  1000
0  Irina  F  2000
1   Mark  M   500
2   Lisa  F  1500]
>>> f2.insert(1,f3)
>>> f2
[    Alan  M  1000
0  Irina  F  2000
1   Mark  M   500
2   Lisa  F  1500,      Alan  M  1000
0  Irina  F  2000
1   Mark  M   500
2   Lisa  F  1500]
>>>
                                                        Ln: 62  Col: 4
```

图 2.17 用 Python 把一个数据转换为列表再加入另一个数据框对象

代码如下。

```
outputpath='C:/Users/Hp/Desktop/f4.csv'
f2[0].to_csv(outputpath,sep=',',index=False)
f4 = pd.read_csv('C:/Users/Hp/Desktop/f4.csv')
f4

outputpath='C:/Users/Hp/Desktop/f5.txt'
f2[1].to_csv('C:/Users/Hp/Desktop/f5.txt',sep='\t',index=False)
f5 = pd.read_csv('C:/Users/Hp/Desktop/f5.txt', sep='\t')
f5
```

应该出现：

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>>
>>> outputpath='C:/Users/Hp/Desktop/f4.csv'
>>> f2[0].to_csv(outputpath,sep=',',index=False)
>>> f4 = pd.read_csv('C:/Users/Hp/Desktop/f4.csv')
>>> f4
    Alan  M  1000
0  Irina  F  2000
1   Mark  M   500
2   Lisa  F  1500
>>>
>>>
>>> outputpath='C:/Users/Hp/Desktop/f5.txt'
>>> f2[1].to_csv('C:/Users/Hp/Desktop/f5.txt',sep='\t',index=False)
>>> f5 = pd.read_csv('C:/Users/Hp/Desktop/f5.txt', sep='\t')
>>> f5
    Alan  M  1000
0  Irina  F  2000
1   Mark  M   500
2   Lisa  F  1500
>>>
                                                        Ln: 80  Col: 4
```

图 2.18 用 Python 从列表中拿出数据形成.csv 和.txt 文件

第五节 撰写脚本语言并且保存

一、撰写脚本语言

目的是为了方便，日后不必重写啊！
另外，也是为了知识交流，
也是为了知识积累，
更是为了知识创造。

二、用 R 撰写脚本语言

如图 2.19 所示

图 2.19 用 R 撰写脚本语言

三、用 Python 撰写脚本语言

如图 2. 20 所示

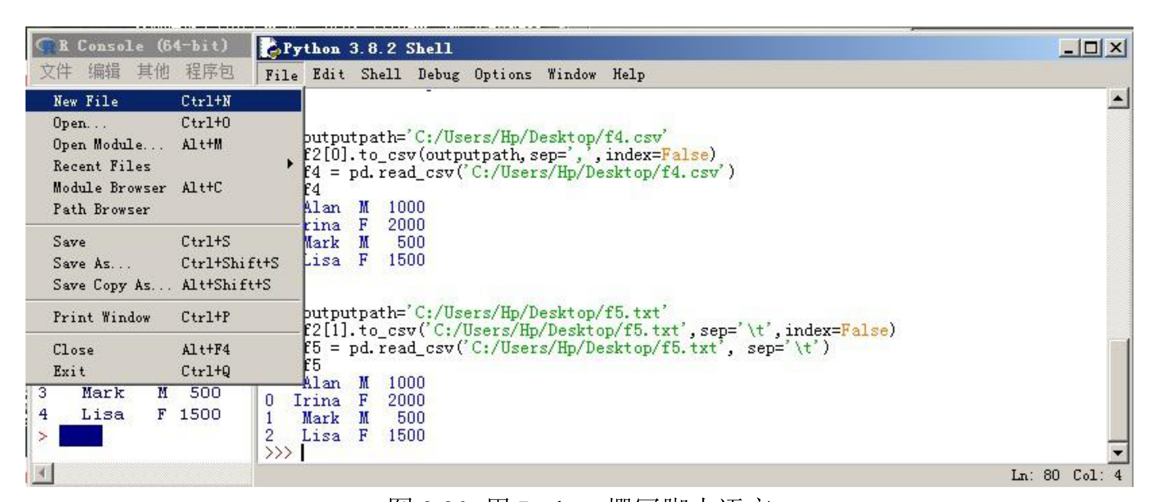


图 2.20 用 Python 撰写脚本语言

第六节 修改脚本语言并且执行

一、修改脚本语言

通常我们会拿到一大长段的代码，此时，除非已经比较熟悉，否则通常我们会采取几种策略。

（一）只运行部分代码，看看能否就像预期想像中那样，发挥该段代码的作用。

（二）修改部分代码的函数、参数或者表达式，借此了解自己是否真正掌握。

（三）更改部分代码，如路径、文件名称、对象的名称，直接拿来就用。

（四）尝试上述所有情况后，开始自己模仿编写一段代码。

（五）自己完全编写一段完全不同，但是结果相同的代码。

二、用 R 执行脚本语言

用鼠标标示后，键入 Ctrl +R 可以只有运行标示的代码。

不用标示，直接在该行代码最后，键入 Ctrl + R 则只运行该行代码。

未命名 - R编辑器

文件 编辑 程序包 帮助

```
x1 <- c("Alan","Irina","Mark","Lisa")
x2 <- c("M","F","M","F")
x3 <- c(1000,2000,500,1500)
f <-data.frame(Person=x1, Sex = x2, RMB = x3)
f

write.csv(f,"C:/Users/Hp/Desktop/f.csv",row.names=FALSE)
f1 = read.csv("C:/Users/Hp/Desktop/f.csv") f1
write.table(f1,"C:/Users/Hp/Desktop/f2.txt",sep='\t',row.names=FALSE)
f2 <- read.table("C:/Users/Hp/Desktop/f2.txt",sep = '\t',header = TRUE)
f2

f3 <- list(f1,f2)
f3
save(f3,file = "C:/Users/Hp/Desktop/f3.rdata")
load(file = "C:/Users/Hp/Desktop/f3.rdata")
f3

write.table(f3[1],"C:/Users/Hp/Desktop/f4.txt",sep='\t',row.names=FALSE)
f4 <- read.table("C:/Users/Hp/Desktop/f4.txt",sep = '\t',header = TRUE)
f4
write.csv(f3[2],"C:/Users/Hp/Desktop/f5.csv",row.names=FALSE)
f5 = read.csv("C:/Users/Hp/Desktop/f5.csv")
f5
```

图 2.17 用 R 修改脚本语言



图 2.18 用 R 执行脚本语言

三、用 Python 执行脚本语言

从本章第二节第三部分到第五节第三部分的所有代码，如图 2.19 所示，并不多。

```
*untitled*
File Edit Format Run Options Window Help
import numpy as np
import pandas as pd
from pandas import DataFrame
data = {'Person': ['Alan','Irina','Mark','Lisa'],
        'Sex': ['M','F','M','F'],
        'RMB': [1000,2000,500,1500]}
f = DataFrame(data)
f

outputpath='C:/Users/Hp/Desktop/f.csv'
f.to_csv(outputpath,sep=',',index=False, header=False)
f1 = pd.read_csv('C:/Users/Hp/Desktop/f.csv')
f1

f1.to_csv('C:/Users/Hp/Desktop/f2.txt',sep='\t',index=False)
f2 = pd.read_csv('C:/Users/Hp/Desktop/f2.txt')
f2

f3 = pd.read_csv('C:/Users/Hp/Desktop/f2.txt', sep='\t')
f3

f2 = [f1]
f2
f2.insert(1,f3)
f2

outputpath='C:/Users/Hp/Desktop/f4.csv'
f2[0].to_csv(outputpath,sep=',',index=False)
f4 = pd.read_csv('C:/Users/Hp/Desktop/f4.csv')
f4

outputpath='C:/Users/Hp/Desktop/f5.txt'
f2[1].to_csv('C:/Users/Hp/Desktop/f5.txt',sep='\t',index=False)
f5 = pd.read_csv('C:/Users/Hp/Desktop/f5.txt', sep='\t')
f5
Ln: 35 Col: 0
```

图 2.19 用 Python 修改脚本语言

```
f = DataFrame(data)
f

outputpath='C:/Users/Hp/Desktop/f.csv'
f.to_csv(outputpath,sep=',',index=False, header=False)
f1 = pd.read_csv('C:/Users/Hp/Desktop/f.csv')
f1

f1.to_csv('C:/Users/Hp/Desktop/f2.txt',sep='\t',index=False)
f2 = pd.read_csv('C:/Users/Hp/Desktop/f2.txt')
f2

f3 = pd.read_csv('C:/Users/Hp/Desktop/f2.txt', sep='\t')
f3

f2 = [f1]
f2
f2.insert(1,f3)
f2

outputpath='C:/Users/Hp/Desktop/f4.csv'
f2[0].to_csv(outputpath,sep=',',index=False)
f4 = pd.read_csv('C:/Users/Hp/Desktop/f4.csv')
f4

outputpath='C:/Users/Hp/Desktop/f5.txt'
f2[1].to_csv('C:/Users/Hp/Desktop/f5.txt',sep='\t',index=False)
f5 = pd.read_csv('C:/Users/Hp/Desktop/f5.txt', sep='\t')
f5
```

图 2.20 用 Python 执行脚本语言



阅读材料

脚本语言

脚本语言需要编译器，因此，就运行效率而言，静态编译语言优于脚本语言：

（1）　在静态编译语言中，C 优于 C++和 Java，又优于 pascal 和 lisp 等。

（2）　在众多脚本语言中，其实 php 优于 ruby 优于 python 优于 perl 等。

因为 Python 无法利用多线程真正并行计算：Python 只有二个核心，却没有一个核快，导致并行运算牺牲复杂性。然而,这要看您如何权衡效率。

（3）　如果快速开发，可以节省人的时间，那么是否机器效率可以低一些？把机器效率问题留到日后计算能力大幅提升时，得到解决。

（4）　Python 是折中方案，可以协作，也可以做到正则化。R 也是同一道理。写出一段代码，Python 是 C 的 1/5 时间，是 JAVA 的 1/3 时间。

本章小结

本章介绍了数据处理的基本流程，我们初步熟悉 R 语言、初步熟悉 Python 编程、初步开始了数据科学。本章主要讲述了以下内容：

(1) 创建数据表格，我们利用数据框（Data Frame）编制了一组信息。这项操作，旨在理解数据表格与数据框的相同相异。

(2) 存取数据表格，把我们所制作的数据框，以.csv 格式，存储在计算机上，成为一组数据集。这项练习，旨在区分数据类型和文件格式的相同相异。

(3) 读写数据集的内容,我们练习在表格(以.csv 文件格式存储的数据)与文本(以.txt 文件格式存储的数据）之间自由切换。这是进行数据处理的常见方式之一。

(4) 在更改数据集格式的练习中，我们利用列表（List）将两个数据框变成一个列表，再由这个列表取出两组数据，各自形成.csv 文件和.txt 文件。这样，多认识了一个不同于数据框的数据类型，并且能够更好地实现格式转换。除此之外，列表能够进行内容抽取和置放的功能，也是常见的数据处理方式之二。

(5) 撰写和保存脚本语言，旨在把之前的工作，予以保存，未来取用时，可以不必重新编写，实现自动化的作用。这是数据处理的通常做法之三。

(6) 读取和修改脚本语言，旨在把之前的工作或者其他人已经做过的工作，拿来测试和修改，进而更有效率地实现数据处理。这是第四项常见方式。

习　题

一、单选题

1、请问 R 和 Python 都有的#符号，代表什么意思？（___）

（A）缺失值（B）空值（C）异常值（D）注释

2、请问<-符号，在 R 语言中，代表什么意思？（___）

（A）赋值（B）定义（C）下标（D）此符号无意义

3、在 R 中，输入 a=1;b=2;c=3 之后，再输入 rm(a,b,c)后，再次输入 a;b;c 的结果为：（___）

（A）1 2 3

（B）错误，找不到对象

（C）6

（D）a b c

4、写入数字和写入字符串有什么不同？（___）

（A）写入字符串需要加上""符号

（B）写入数字需要加上<-和（）符号

（C）写入数字需要加上{}

（D）写入字符串要用 assign()函数

5、如果 wahaha 的数据类型是 data.frame 所以 wahaha[3,2]是指（___）
（A）wahaha 的第三行第二列
（B）wahaha 的第三列第二行
（C）wahaha 出现三次的 a 和出现两次的 h
（D）wahaha 的第三个位置的 h 和第二个位置的 a

二、 多选题

1、建立一组基于【数据框】形式的数据集，应该采用的步骤是（___）
（1）赋值给数据框
（2）建立向量以及变量
（3）把数据框写成需要的文件格式
（4）存储并且删除工作环境的数据框，然后调用文件进行测试；

三．问答题

1、我们在 R Console 和 Python Shell 各自输入 w=5 然后关闭视窗；
接着，开启 R 和 Python 时，在 R 和 Python 里，都输入 w 之后；
在 R 显示数字 5 而在 Python 里显示错误提示 Traceback 为什么？
2、请在 R Console 输入 contributors() 然后告诉我们 R 为什么称为 R 而不是其他字母？
3、请在 Python Shell 输入 import this 然后告诉我们什么是“编程之禅”？

四．课后作业

第一章的课后作业，是下载十篇论文，这一章的课后作业，是针对这十篇论文进行解读。

1.作业释疑：解读什么呢？
数据来源，即：该论文的数据，主要来自哪里，例如：
（1）作者自产数据集（如问卷调查）、实验（如仪器记录）得来；
（2）作者取自数据库，导出数据得来；
（3）作者取自其他研究（其他论文、其他数据论文、其他科研成果）得来；
（4）该论文披露数据；（5）...；（6）...；....等。

2.作业释疑：提交的.csv 文件，应该包括那些内容？
编号、论文题目、数据来源；如下：

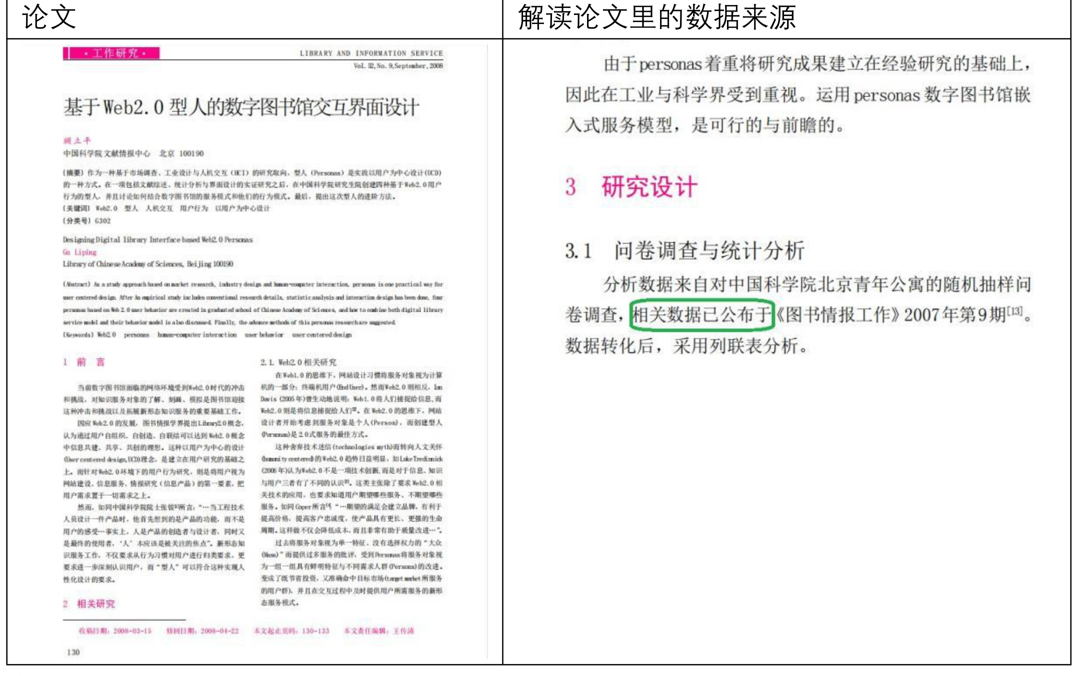
编号	论文题目	数据来源	数据来源说明
01	基于 Web2.0 型人的数字图书馆交互界面设计	作者取自其他研究	顾立平. Web2.0 用户行为调查研究-以中科院学生使用社会软件为例[J]. 图书情报工作,2007,51(9):100-103.
02	Introduction to information seeking behavior - A review of literature and field practice directions	作者取自数据库	LISA 文摘索引数据库

03	A discussion of the emotive delement of knowledge service practice: An empirical study at the Chinese Academy of Sciences	作者自产数据集	问卷调查
04	基于 Web2.0 用户信息检索行为的交互设计：后设分析与问卷调查研究	该论文披露数据	问卷调查
…	…	…	…

3.作业释疑：能否进一步示范？

编号 01

论文	解读论文里的数据来源



·工作研究·　LIBRARY AND INFORMATION SERVICE

Vol. 52, No. 9, September, 2008

基于Web2.0型人的数字图书馆交互界面设计

顾立平

中国科学院文献情报中心　北京 100190

〔摘要〕作为一种基于市场调查、工业设计与人机交互（HCI）的研究取向，型人（Personas）是实践以用户为中心设计(UCD)的一种方式。在一项包括文献综述、统计分析与界面设计的实证研究之后，在中国科学院研究生院创建四种基于Web2.0用户行为的型人，并且讨论如何结合数字图书馆的服务模式和他们的行为模式。最后，提出这次型人的进阶方法。

〔关键词〕Web2.0　型人　人机交互　用户行为　以用户为中心设计

〔分类号〕G302

Designing Digital library Interface based Web2.0 Personas

Gu Liping

Library of Chinese Academy of Sciences, Beijing 100190

〔Abstract〕As a study approach based on market research, industry design and human-computer interaction, personas is one practical way for user centered design. After An empirical study includes conventional research details, statistic analysis and interaction design has been done, four personas based on Web 2.0 user behavior are created in graduated school of Chinese Academy of Sciences, and how to combine both digital library service model and their behavior model is also discussed. Finally, the advance methods of this personas research are suggested.

〔Keywords〕Web2.0　personas　human-computer interaction　user behavior　user centered design

1 前　言

当前数字图书馆面临的网络环境受到Web2.0时代的冲击和挑战，对知识服务对象的了解、刻画、模拟是图书馆迎接这种冲击和挑战以及拓展新形态知识服务的重要基础工作。

因应Web2.0的发展，图书情报学界提出Library2.0概念，认为通过用户自组织、自创造、自联结可以达到Web2.0概念中信息共建、共享、共创的理想。这种以用户为中心的设计……

2 相关研究

2.1 Web2.0相关研究

在Web1.0的思维下，网站设计习惯将服务对象视为计算机的一部分：终端机用户(End User)。然而Web2.0则相反，Ian Davis (2005年)曾生动地说明：Web1.0将人们捕捉给信息，而Web2.0则是将信息捕捉给人们。在Web2.0的思维下，网站设计者开始考虑到服务对象是个人(Person)，而创建型人(Personas)是2.0式服务的最佳方式。

这种舍弃技术迷信(technologies myth)而转向人文关怀……

130

由于personas着重将研究成果建立在经验研究的基础上，因此在工业与科学界受到重视。运用personas数字图书馆嵌入式服务模型，是可行的与前瞻的。

3 研究设计

3.1 问卷调查与统计分析

分析数据来自对中国科学院北京青年公寓的随机抽样问卷调查，相关数据已公布于《图书情报工作》2007年第9期[13]。数据转化后，采用列联表分析。

编号 02

论文	解读论文里的数据来源

Introduction to information seeking behavior—A review of literature and field practice directions

Li-Ping KU

Library of City University of Hong Kong, Hong Kong SAR, China

Received Feb. 3, 2009
Revised Dec. 8, 2009
Accepted Mar. 3, 2010

Abstract The article tries to discover the major authors in the field of information seeking behavior via social network analysis. It is to be accomplished through a literature review and also by focusing on a graphic map showing the seven most productive co-authors in this field. Based on these seven authors' work, five probable research directions about information seeking behavior are discerned and presented.

Keywords Social network analysis, Library and information science database, Information seeking behavior, User behavior

1 Introduction

As important reference resources of library services, library information systems, library workflows, library websites, and the study of information seeking behavior are some of the main research areas in library science.

In the physical library building, when a person uses a library or communicates with library staff, a series of encountering phenomena takes place such as follows:

access→contacting with the media→technology and computer→information→borrowing items→returning items→independent learning→pay fine, suggestion box system and the layout of the library→librarians' professional expertise→catalog, index, classify, rules and regulations[1-3].

However, the role of the Internet has become increasingly important for both educators and students to obtain useful information. The model of readers' information seeking behavior considers not only their behavior in the library building but also the general information environment in their living and work surroundings. It takes into account the rapid development of the information environment, in which learners live in an interactive communication village of common knowledge sharing and common knowledge creation. These unprecedented new challenges

Correspondence should be addressed to Ku Li-Ping (E-mail: lipingku@cityu.edu.hk).

Both Question 1 and Question 2 are addressed in the "Result" of social network analysis in this paper, but Question 3 is presented in the "Discussion" such on this paper. The conclusion of this paper includes the summary of practice directions, study limitations and future research.

3 Research method and process

The research method is a social network analysis: A kind of data mining approaches. There are four steps: data collection, data clearance, data analysis, and graphic interpretation.

1st step. Data collection is to retrieve articles from LISA, and the process is to input "information seeking" as the keywords for search. There were 1,889 items downloaded from that database on September 1, 2008.

2nd step. Data clearance is to write a computer program (using the JAVA platform) in order to transfer data format. The format ".txt" used in LISA database was transformed into another form of format ".txt" which can be processed by the Ucient.

3rd step. Data analysis is to use Ucinet to calculate each author's number of publications, their cooperative number, and the co-authors' relationship.

4th step. Graphic interpretation is to present the map which includes some network nodes (authors) and network lines (cooperative relationship). By simplifying the map of social network relationship, the final graph shows more directly and clearly the key author's location and relationship number.

Using these 4 steps of the above mentioned research process, the research result shows in two graphs: One is the whole map of the co-authors' relationship, the other is the simplified map of the key authors' strength (the number of published articles) and influence (the social relationship network).

4 Result

Based on the above process, a graph is drawn below for the purpose of elucidation. There are two maps of the social network relationship in the research field of information seeking behavior.

The first map is shown in Fig. 1. It answers the first research question about what the social network relationships of those authors look like. It shows that there are three main groups (in the middle of the graph) and many other occasional authors (on the left list of the picture). For a better understanding of the key authors, their work team and their social relationship network, Fig. 2 is developed for simplification based on Fig. 1.

编号 03

论文	解读论文里的数据来源

A discussion of the emotive element of knowledge service practice: An empirical study at the Chinese Academy of Sciences

Li-Ping Ku[1,2]

[1] National Science Library, Chinese Academy of Sciences, Beijing 100190, China
[2] Library of City University of Hong Kong, Hong Kong SAR, China

Received Nov. 11, 2009
Revised Dec. 29, 2009
Accepted Jan. 30, 2010

Abstract With the arrival of the information age, research activities focused on the practice and approaches of knowledge services are on a marked increase as evidenced in the publications of social sciences. According to a social network analysis on knowledge service related literature, it reveals that information and knowledge workers often fail to take such an important element as the functional role of an emotive engagement into consideration in their study of knowledge services. It has increasingly become an issue of high profile with the rapid development of digital libraries and their web-based knowledge services in China and anywhere else in the world. In order to have a clearer understanding about issues involved in knowledge servicing so as to maximize the effectiveness and efficiency of digital libraries in their knowledge service performance, the author has conducted surveys for seven times on the online information seeking behavior of graduate students at the Chinese Academy of Sciences with such research methods as questionnaires, interviews and natural observations during September 2006-June 2009. The research result has showed the emotive element has an important role in the user's information seeking behavior and in knowledge services practice. Therefore, knowledge services rendered may be more effective by adding the emotiveness-oriented communication element into such practice. This paper recommends that such an emotiveness-oriented communication approach should be carefully studied and factored into libraries' knowledge services practice.

Keywords Emotive engagement, Personas, Personnel profile, Information seeking behavior, Digital library, Knowledge innovation, Knowledge service

1 Introduction

Along with the advancement of the Internet technologies and the development of Information Science, more and more scientists and engineers begin to realize the importance of the knowledge service industry. For supporting the reading public's information demands, the library and information science professionals perform the value-added information services, which includes 1) Information organization;

Correspondence should be addressed to Ku Li-Ping (E-mail: lipingku@cityu.edu.hk).

Table 1 Process of investigation during Jan. 2007 to Jun. 2009

Period	Survey	Sample object	Sampling method	Total sample	Valid sample	Notes
From Jan. 7 to Jan. 20, 2007	Questionnaire	CAS Student Apartments of Zhongguancun	Systematic sampling	400	263	Preliminary
From Jan. [illegible] to Jan. 27, 2007	Questionnaire	CAS Student Apartments of Zhongguancun	Systematic sampling	500	358	Preliminary
From May [illegible]1 to May 31, 2007	Questionnaire	Students of CAS, Taiwan University and National Yunlin University of Science	Simply sampling	159	159	Preliminary
From Jun. 9 to Jul. 20, 200[illegible]	Questionnaire	Graduate School of CAS (Beijing)	Stratified random sampling	3,000	498	Basic survey
From Oct. [illegible]0 to Nov. 1, 200[illegible]	Questionnaire	The previous survey	Purposive sampling	300	155	Continuous
From Nov. [illegible]5 to Nov. 25, 2008	Interview	The previous survey	Purposive sampling	40	36	Continuous
From Jan. 5 to Jun. 5, 2009	Observation	The previous survey	Purposive sampling	16	8	Continuous

The fifth survey was through purposed random samplings to collect the answers from respondents and non-respondents of the fourth questionnaire. It added fifteen additional closed questions and five open-ended questions to enquire about their library service innovations. The proportion of returned questionnaires was that the respondents accounted for 20% and the non-respondents made up 80%.

The sixth survey followed up the fifth questionnaire and it also used purposed random samplings to collect the interview handwriting data in the same samplings. The proportion of returned questionnaires between the respondents and the non-respondents are 50 versus 50 percent. The goal of this survey was to understand the reasons and details of the returned answers of the fourth and fifth questionnaires.

The seventh survey was a natural observation during the six-month survey period for more detailed understanding of the interviewed respondents. These continuous investigations included the conduction of in-depth interviews for three times and of correspondence via emails using a question-answer format for four times.

4 Results

4.1 The first survey finding: More entertainment feature attractions than practical educational opportunities in the Web 2.0 information environment

The results of this survey showed that the mobile phone is an important platform for Web2.0 users. 55% students selected the informal way to get desktop softwares, it appears that the free software is more popular than the software's brand name.

编号 04

论文	解读论文里的数据来源

总第 129 期
2009 年 5 月

图书·情报·知识

No. 129
May, 2009

基于 Web2.0 用户信息检索行为的交互设计：后设分析与问卷调查研究

顾立平
(中国科学院研究生院，北京，100049；中国科学院文献情报中心，北京，100190)

[摘要] 设计交互式数字图书馆依赖于用户特征、行为、工作习惯和需求的描述。此过程有两个部分：用户行为调查报告的后设分析，在此分析基础上，另一个是对中国科学院研究生院的博硕士生进行的问卷调查。调查结果对博客、及时通讯、社会网络软件、社会标签和个性化图书馆，各种结合数字图书馆的可能方式，带来更多讨论。结论建议哪些数字图书馆功能对用户而言实属必要。

[关键词] Web2.0 用户行为 后设分析 问卷调查 信息检索行为
[中图分类号] G354 [文献标识码] A [文章编号] 1003-2797(2009)03-0026-09

[Abstract] Designing interactive digital library depends on describing user's character, behavior, working habit and need. This process has two parts: one is meta-analysis on survey reports of users' behavior; based this analysis the other is questionnaire survey to PhD and post-graduate students of Graduate University of Chinese Academy of Sciences. The survey results bring more discussion on many probable ways of digital library with Blog, Instant Messanger, Social Network Software, Social Tag and Personal Library. Conclusion suggests which digital library functions for users necessary are.

[Key words] Web2.0 User behavior Meta-analysis Questionnaire Information seeking behavior

1 前言

在设备升级、价格下降、人人参与 Web2.0 的时代，尊重用户、融入用户体验、节省用户培训时间的数字图书馆，才能够赢得用户的喜好与利用。而这需要将编程与设计分家[1]。过去为考虑储存、运行与传输限制，而以编程为核心，混合功能设计的作法，造成了许多实际应用上的接触困难[2]。有理由相信，由于产品设计不良，而不是技术或培训问题，才是造成用户不愿意接触数字图书馆的原因[3-4]。

从理解用户行为，转化为交互设计，再到编程的制作过程，方是为用户打造适合他们的数字图书馆。然而，摆在眼前的困难是：对 Web2.0 用户而言，利用数字图书馆检索信息时，需要那些功能？那些新技术？那些新服务？对节省成本、注重效率、热心服务的数字图书馆而言，急需 Know-How！

国外 OCLC、ProQuest 和 PEW 进行了许多有意义的用户研究，但是并不以数字图书馆的交互设计为目的，而研究其知识服务对象的行为研究。国内的 CNNIC 对中国网民、清华大学图书馆对大学生、中科院文献情报中心对研究生都进行过不同规模的用户/读者调查。《数字图书馆论坛》甚至译介 OCLC 报告。上述多半调查"已经发生过"的行为，而很少注意"将要或可能发生"的用户行为。

对数字图书馆设计而言，依然存在的实际难题是：如何有效的"预知"三至五年以后的用户需求？一个软件最快的开发周期无非是两年，而一个用户调研最快也需要一年的工作时间，但是计算机技术每半年一个周期。如何比用户提前知道图书馆该做什么服务？其实也是对图书馆员的素质要求。本文利用国内外已有的研究报告，进行后设分析(meta-analysis)，筛选出哪些是已知的，哪些是未知的。再对未知的部分，进行问卷调查，以提供更为深入的

[基金项目] 本研究受中国科学院港澳台留学生奖学金的资助。
[作者简介] 顾立平，男，1978 年生，博士研究生，发表论文 4 篇。

续表 1

作者单位	调查主题	报告内容与结论(节录)	地点	对象	方法	样本
PEW，2008[14]	2008 全球态度调查·中国篇	1. 64 %拥有个人电脑 2. 62 %使用网络 3. 75 %使用 Email 4. 新闻来源：电视 88 %而网络只占 5 %	中国(北京)	18 岁以上成人	面对面问卷调查	3212
ALA，2007-08[15]	学习学生：Rochester 大学本科生研究	1. 学生：找优秀论文要在图书馆里面，写优秀论文要在图书馆外面。 2. 学生的学习时间是图书馆的正常下班时间。	美国	校内本科学生	面访访谈观察	1415

(3) 理论上争议的结果，研究推论限制的条件。信息检索行为模式的优点在于具有观察整体用户行为的指标和内容，缺点在于不能很好地解释当前网络用户的多样性和网络工具的多选择性。Web2.0 共享模式的优点在于符合一般网络用户行为，缺点在于难以说明如何适用于科学、教育与学习[19]。择优防劣，正好说明了"Web2.0 信息检索行为"应该用什么理论基础，观察什么网络现象，以及避免什么可能存在的研究设计陷阱。

(4) 对初探性研究的可能方向。综上所述，对下一代数字图书馆用户行为的探索研究，重点不在图书馆里面，而在 Web2.0 世界[20-21]。以 ISBM 来梳理对 Web2.0 用户行为的调查，可以得到较为深入的解释与挖掘新的问题。阶段性的调查结果有两种意义：对传统信息检索行为模型提出新的见解；以及对照当前种种网络用户行为调查的结果，提出数字图书馆下一阶段的因应之道。

3 问卷调查研究设计

根据"2 后设分析"的结论，进行以"数字图书馆的交互设计"为目的的用户研究，设计问卷调查如下：

(1) 研究方法与工具。利用 SAS 8.1 版进行数据分析，采用描述统计、方差分析与单样本 T 检验。(2) 抽样设计。研究对象：中国科学院博硕士研究生；抽样对象：2008 年第二学期中国科学院北京地区在读博硕士研究生；样本：填写问卷有效而且被回收的人员。

(3) 问卷设计。共计 82 项，包括 7 个部分：个人信息、Web2.0 态度、Web2.0 行为、Web2.0 认知、及时通讯与博客、社会书签与社会网络(问卷标题：网络社群参与意愿)、学习习惯等。

(4) 数据采集。根据协办单位：中国科学院研究生院提供的学生数据，进行分层随机抽样问卷调查。以中国科学院文献情报中心的名义，自 2008 年 6 月 9 日至 2008 年 7 月 20 日发放与回收问卷。邮寄发放 3058 份，回收 501 份，有效问卷 497 份。

4 问卷调查结果

4.1 基本数据(见表 3)

表 3 基本数据(年龄层、专业两项，经过数据归整)

性别		男		女	
人数		324		173	
比例		65.2		34.8	
年龄层	23 岁以下 1985	24～25 岁 1984～1983	26～27 岁 1982～1981	28～29 岁 1980～1979	30 岁以上 1978
人数	52	171	170	53	51
比例	10.5	34.4	34.2	10.7	10.3
教育程度		博士		硕士	
人数		349		148	
比例		70.2		29.8	
专业	基础科技	生命科学	环境资源	高新技术	其它
人数	154	124	78	107	34
比例	31	24.9	15.7	21.5	6.8
网龄	1～3 年	4～6 年	7～10 年	10 年以上	
人数	5	158	286	48	
比例	1	31.8	57.5	9.7	
每周上网	不确定	少于一天	1～3 天	4～6 天	每天
人数	4	2	14	45	432
比例	0.8	0.4	2.8	9.1	86.9
信度检验：项目数为 98；Cronbach's Alpha 为 0.883；大于 0.5，信度合格。					

4.2 相关系数卡方检验

利用"性别"、"年龄"、"教育程度"、"专业"四个定类变项，分别假设它们与其他 94 个变量互独立，查 X^2 分布表，取检验显著性水平 $\alpha = 0.05$；对照其自由度，比较其方差值与临界值。结论：固然有零星的结果推翻假设，比如：性别和社群之间存在若干关系；但是绝大部分并没有推翻原先假设：两变量间相互独立。

4.3 受访者年龄

根据单样本 T 检验。在 95 %信赖区间内，本次调查的受访者出生年在 1981.53 到 1982.02 之间。换言之，本次调查代表年龄在 27 岁半到 28 岁之间的研究生。

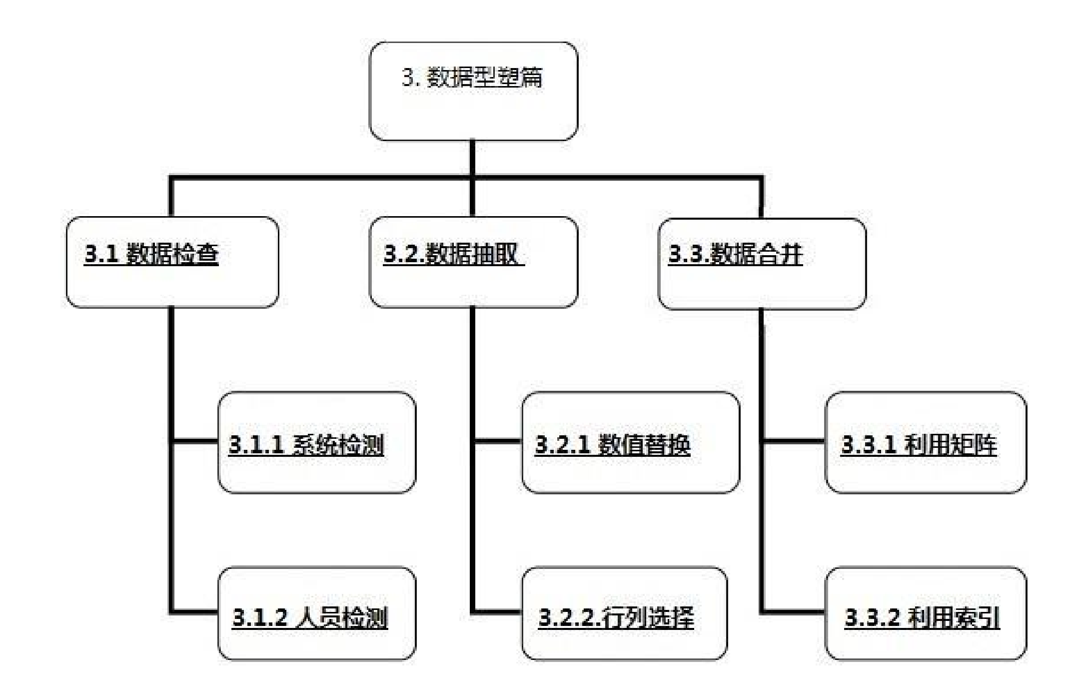
4.4 博客(见表 4)

第三章　数据型塑篇

学习目标

理解缺失值、空白值、空值的异同；
掌握检查一个测数据集的方式；
理解数据框及其行、列、数值等；
掌握形成新的数据集的方式；
了解数据框、矩阵、索引的不同形式；
掌握操作矩阵和索引的方式。

知识结构



导入案例

我们在上一章制作了 4 个人的三组信息，为了本章以及日后章节的学习，我们把行（样本、人数）扩张到 10 行，把列（变量）扩张到 5 行。代码示例，如下：

```
x1 <- c("Alan","Irina","Mark","Lisa","Swan","Jack","Tom","Mary","Joe","Bonnie")
x2 <- c("M","F","M","F","F","F","M","F","M","F")
x3 <- c(1000,2000,500,1500,800,1500000,1200,1400,900,1100)
x4 <- c(1,2,4,3,5,6,7,8,9,10)
x5 <- c("D","D","D","D","M","B","B","B","M","M")
 # 其中 Jack 的性别，写为 F。
```

```
  # 其中 Jack 的 RMB，写为 1500000 明显高于其他人。
  # 其中 Mark 和 Lisa 的次序，分别为 4 和 3 的情况。
f <-data.frame(Person=x1, Sex = x2, RMB = x3, Order = x4, Edu= x5)
f
```

上述代码，我们可以键入 R Console 后，一条条运行，即第二章第一节的方法；或者，写在.r 文件，接着一次运行，即第二章第五节的方法。

所示结果，如下：

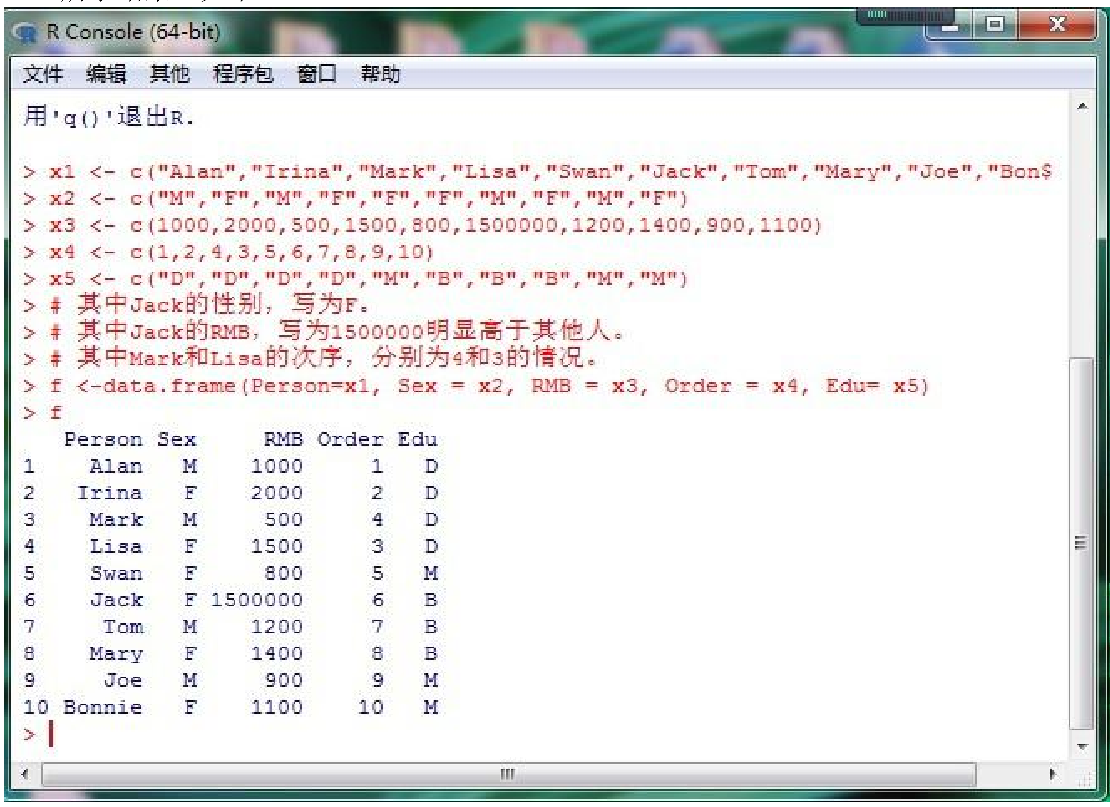


图 3.1 新的试验数据集（R）

同理，我们也可使用 Python 建立同样的模块.py 文件，或者，数据集.csv 文件。

```
import numpy as np
import pandas as pd
from pandas import DataFrame
data = {'Person': ["Alan","Irina","Mark","Lisa","Swan","Jack","Tom","Mary","Joe","Bonnie"],
    'Sex': ["M","F","M","F","F","F","M","F","M","F"],
    'RMB': [1000,2000,500,1500,800,1500000,1200,1400,900,1100],
    'Order': [1,2,4,3,5,6,7,8,9,10],
    'EDU': ["D","D","D","D","M","B","B","B","M","M"],
        }

f = DataFrame(data)
f
```

我们把上述代码，写为.py 文件（例如，命名为 3.1.py）接着运行。在 Python Shell 里我们再键入 f 把对象调用出来，放在 Python Shell 的环境里。所示结果，如下：

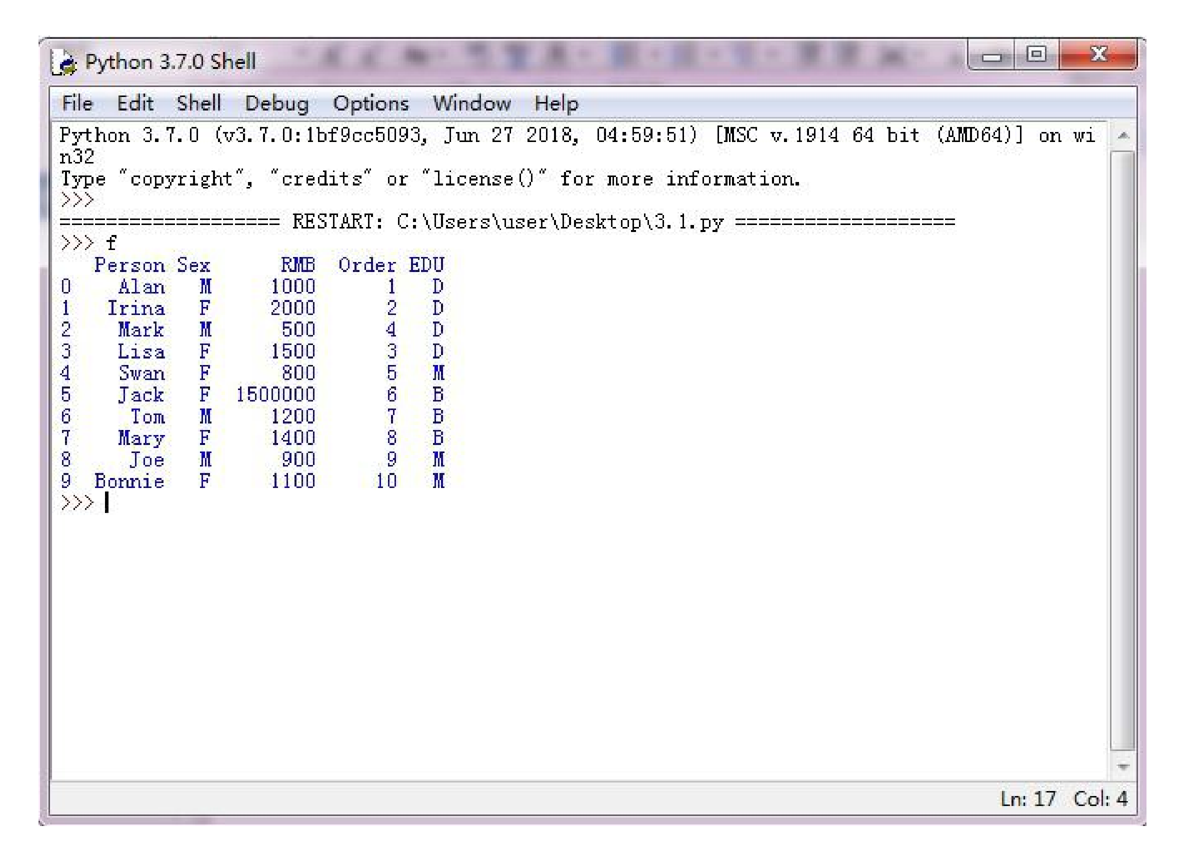

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:\Users\user\Desktop\3.1.py ===================
>>> f
    Person Sex      RMB  Order EDU
0     Alan   M     1000      1   D
1    Irina   F     2000      2   D
2     Mark   M      500      4   D
3     Lisa   F     1500      3   D
4     Swan   F      800      5   M
5     Jack   F  1500000      6   B
6      Tom   M     1200      7   B
7     Mary   F     1400      8   B
8      Joe   M      900      9   M
9   Bonnie   F     1100     10   M
>>>
```

图 3.2 新的试验数据集（Python）

请注意：此处，我们故意输入一些不正常的数值，比如 Jack 的性别（Sex），输入为 F（应该为 M），比如 Jack 的 RMB 明显超过其他人，这样，便于我们学习本章内容。

假如我们有成千上万条数据，当然不能用我们刚刚做的那样，一条一条数据输入，我们应该采取 3.3 数据抽取和 3.4.数据合并的策略。目前，我们首先考虑的是拿到一笔数据之后，怎么进行检查，以及怎么处理数据集里面那些比较棘手的数据。

第一节 数据检查

一、系统检测

现在我们拿到这笔数据集了，我们把性别（Sex）变量的其中一笔数据抹掉，会有两种方式。第一种：空值。

```
x2 <- c("M","F","M","F","F", ,"M","F","M","F")
# 第六个位置，没有填写任何东西。
# 这样，R 会告知 【6 元素】是空的
```

第二种：缺失值。

```
x2 <- c("M","F","M","F","F", " ","M","F","M","F")
# 第六个位置，有" "表示该位置也是字符串，但是里面没有具体的字符串内容。
# 这样，R 并不会认为有什么问题。
```

两种情况的比较，请见：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> x5 <- c("D","D","D","D","M","B","B","B","M","M")
> # 其中Jack的性别，写为F。
> # 其中Jack的RMB，写为1500000明显高于其他人。
> # 其中Mark和Lisa的次序，分别为4和3的情况。
> f <-data.frame(Person=x1, Sex = x2, RMB = x3, Order = x4, Edu= x5)
> f
   Person Sex     RMB Order Edu
1    Alan   M    1000     1   D
2   Irina   F    2000     2   D
3    Mark   M     500     4   D
4    Lisa   F    1500     3   D
5    Swan   F     800     5   M
6    Jack   F 1500000     6   B
7     Tom   M    1200     7   B
8    Mary   F    1400     8   B
9     Joe   M     900     9   M
10 Bonnie   F    1100    10   M
>
> x2 <- c("M","F","M","F","F", ,"M","F","M","F")
Error in c("M", "F", "M", "F", "F", , "M", "F", "M", "F") : 6元素是空的
>
> x2 <- c("M","F","M","F","F"," ","M","F","M","F")
> x2
 [1] "M" "F" "M" "F" "F" " " "M" "F" "M" "F"
```

图 3.3 空值与缺失值的比较（R）

同理，我们也可以在 Python 里试试，结果也会相同。系统会仔细检查是否存在空值，在 Python 里，是 invaild syntax 无效语法。请见：

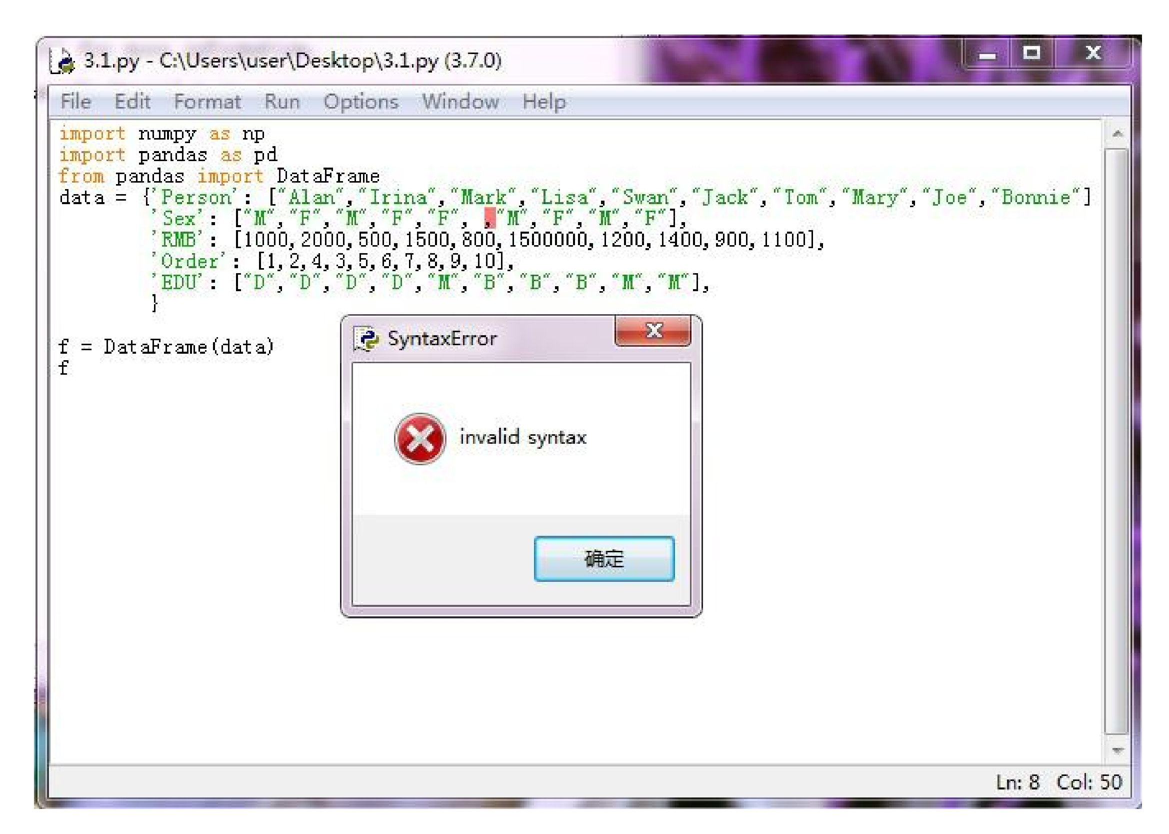


图 3.4 空值被视为语法无效（Python）

用 Python 进行缺失值的修改，在字符串的方面，要用 np.nan 表示缺失值。如果仅仅是“ ”中间没有填写任何数值，那么仍然会被认为是具有字符串的内容。

```
data = {'Person': ["Alan","Irina","Mark","Lisa","Swan","Jack","Tom","Mary","Joe","Bonnie"],
     'Sex': ["M","F","M","F","F",np.nan,"M","F","M","F"],
     'RMB': [1000,2000,500,1500,800,1500000,1200,1400,900,1100],
     'Order': [1,2,4,3,5,6,7,8,9,10],
     'EDU': ["D","D","D","D","M","B","B","B","M","M"],
        }

f = DataFrame(data)
```

这也表示了 R 和 Python 设计理念的不同：R 是基于统计学家的角度，告诉我们，元素 6 是空值，而 Python 是基于计算机学家，告诉我们，语法错误。

此外，我们还可以知道另一件事：空值和缺失值的不同。空值是一种语法错误，应该给出元素但是什么都没有的错误，缺失值是一种没有给出具体数值，但是语法上没有错误的情况；因此，无论是 R 还是 Python 都不会在系统里（R Console 和 Python Shell）提示和告知我们存在缺失值。

换言之，我们进行数据检查，需要用另外的一些方式，来找到缺失值。

在此之前，我们可以先核实一下 R 和 Python 在缺失值的情况下的相同点。

R Console (64-bit)

文件 编辑 其他 程序包 窗口 帮助

```
1     Alan   M    1000     1   D
2    Irina   F    2000     2   D
3     Mark   M     500     4   D
4     Lisa   F    1500     3   D
5     Swan   F     800     5   M
6     Jack   F 1500000     6   B
7      Tom   M    1200     7   B
8     Mary   F    1400     8   B
9      Joe   M     900     9   M
10  Bonnie   F    1100    10   M
>

> f <-data.frame(Person=x1, Sex = x2, RMB = x3, Order = x4, Edu= x5)
> f
   Person Sex     RMB Order Edu
1     Alan   M    1000     1   D
2    Irina   F    2000     2   D
3     Mark   M     500     4   D
4     Lisa   F    1500     3   D
5     Swan   F     800     5   M
6     Jack     1500000     6   B
7      Tom   M    1200     7   B
8     Mary   F    1400     8   B
9      Joe   M     900     9   M
```

图 3.5 没有缺失值和有缺失值的数据框的比较（R）

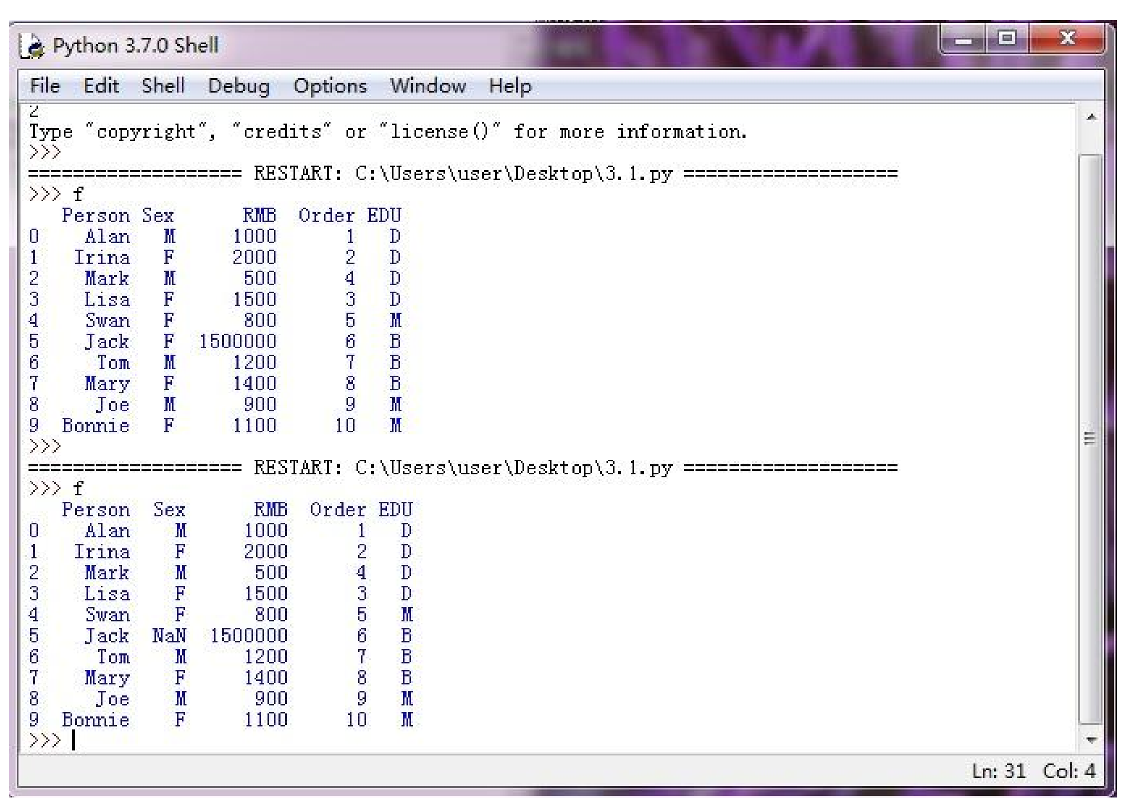


图 3.6 没有缺失值和有缺失值的数据框的比较（Python）

如果您很细心的话，还可以发现图 3.5 里，我们写了一行：

```
f <-data.frame(Person=x1, Sex = x2, RMB = x3, Order = x4, Edu= x5)x2 <- c("M","F","M","F")
```

这是因为 R Console 里，会保存之前运行过的对象。因为我们之前建立了一个 f 对象，以数据框的形式，里面具有 x1,2...x5 的信息。那么，我们虽然在 R Console 里，更改了 x2 的情况（把第六位置的元素，空下，成为缺失值），但是 f 这个对象仍然是包括原来那个 x2（第六位置的元素没有空下）。

所以，我们需要重新建立一个 data.frame 把后来经过修改的 x2 放进去，这样，才能看到修改后的 f 打印出来，在第 6 行第 2 列是没有显示任何数值的。

如果我们在.r 文件里，直接修改，直接运行，就不用再在 R Console 里再写一次 data.frame 的代码了。如同我们在.py 文件里面直接更改，然后运行的那样。

上面这段话，有点拗口，实际上，如果您操作过，就远比看着这串文字去想是什么意思，来得更为容易理解。

二、人员检测

从前面可知，系统会告诉我们空值，因为空值代表着语法错误。但是，缺失值（看不到有任何数值的情况）以及异常值（明显和其他同列数值，差异较大，例如大家都是 RMB 月收入 1000 元上下，但是 Jack 是 150000 之多），系统不会自动提示我们。

所以，数据量少，那么我们还可以凭借肉眼一笔笔看，但是数据量大的时候，我们就需要借由其他方式了。

如下代码：

```
summary(f)
```

是的，您没看错，就一行。见下图

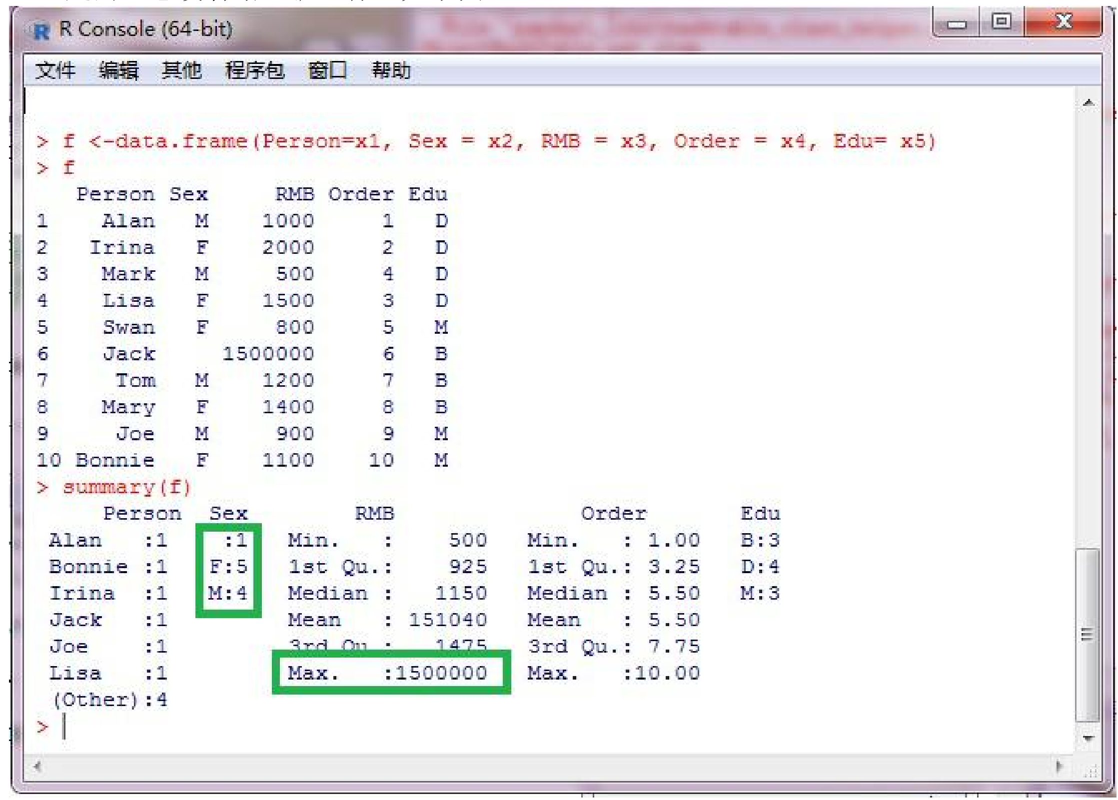


图 3.7 用 R 检查缺失值和异常值

以上，我们看到 Sex 列，有一个：1 以及 F：5 和 M：4 代表着，有 1 个没有数值，数值为 F 的有 5 个，数值为 M 的有 4 个。

换言之，有 1 位不知性别，有 5 位是女性（F），有 4 位是男性（M）。

稍微对比一下 Edu 那列，即可知道，本科（B）、博士（D）、硕士（M）都有统计出来，各自出现的次数。

再看 RMB 那列，最大值 1500000 明显高过平均值 10 倍以及中位数 150 倍。

所以，我们可以知道，在 Sex 那列，有缺失值，在 RMB 那列，有异常值。我们看出来的前提是，具有统计知识以及知道如何阅读 R 的报表。

接下来，我们使用 Python 进行数据检查：

```
f.describe()
  # Python 的 Pandas 的函数 describe( )只有连续型数据的描述统计。
  #顺序（Order）变量的数值（1,2...10）被 Python 当做是连续型数据，可以忽视。

f[f['RMB']>100000]
  #根据观察，我们发现过滤异常值的方法，可以是 100000 以上，所以这么写。
  #如何判断异常值，还有很多方式，根据不同的方式，写的 Python 代码不一样。

f .isnull().any()
  # 在 f 里找寻任何有缺失值的列。
  # 如果没有加上.any()的话，会出现数据框的每个行列，所对应的真值（T）假值（F）
```

应该出现：

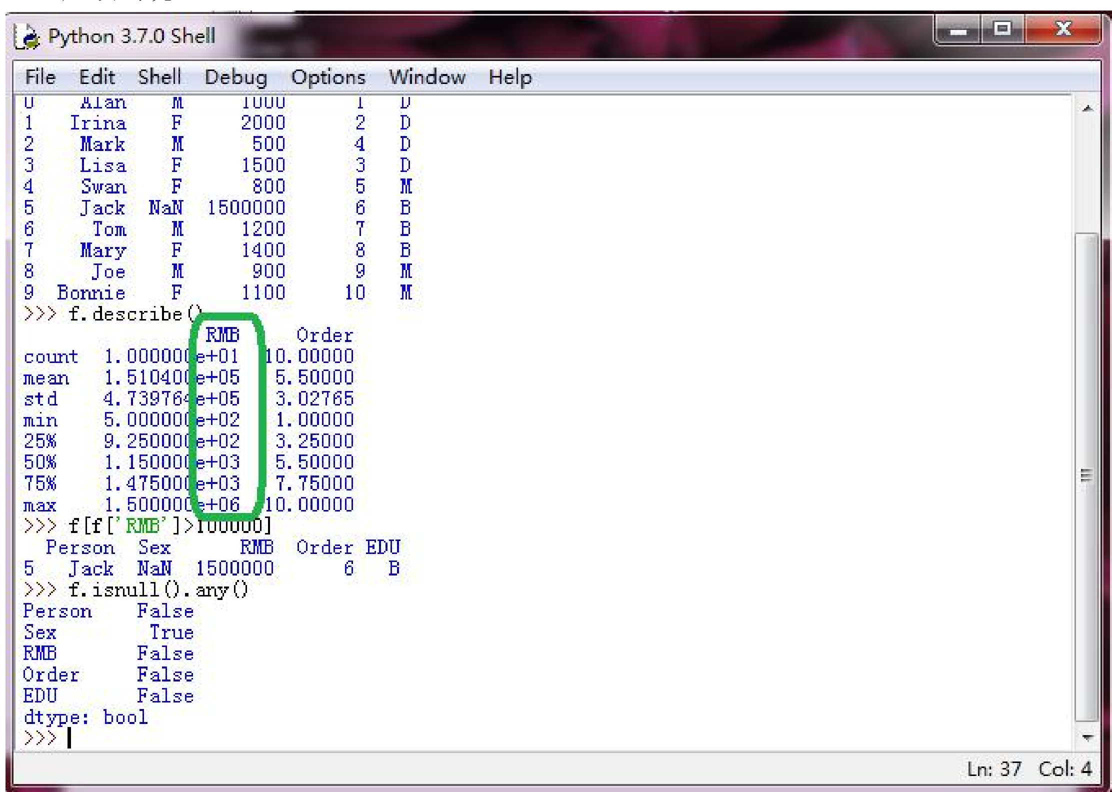


图 3.8 用 Python 检查缺失值和异常值

如图 3.8 我们看到 RMB 的各项统计描述，其中最大值 max 有六次 10 进位的量；而 Order 的各项统计描述可以不看，因为我们知道 Order 是 1,2,...10 的顺序排列，其数字 1,2,...10 没有计算上的意义。

接着，我们查看到了，在 Sex 变量，有缺失值 null 为真（True）的情况，其他变量都是假（False），这代表了，在 Sex 列，存在缺失值，而其他列，并没有。

那么，我们的检查工作，算是完成了，接下来就是怎么进一步解决它。

第二节 数据抽取

一、数值替换

我们首先先把 Jack 从女性（F）变回男性（M）。这是我们刚刚无论从空值、缺失值、还是异常值，都没有办法检测到的内容。

首先，我们用 R 把数据框（Data Frame）转为矩阵（Matrix）形式；接着，就容易在矩阵里，找到对应的行列，进行赋值（assign）的动作，把 F 用 M 替换掉。代码如下：

```
f<-as.matrix(f)
# 把 f 变成矩阵

f[6,2]<-'M'
# 在第 6 行 2 列，写入 M 这个字符串
```

应该出现：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> f<-as.matrix(f)
> f
      Person   Sex RMB       Order Edu
 [1,] "Alan"   "M" "   1000" " 1"  "D"
 [2,] "Irina"  "F" "   2000" " 2"  "D"
 [3,] "Mark"   "M" "    500" " 4"  "D"
 [4,] "Lisa"   "F" "   1500" " 3"  "D"
 [5,] "Swan"   "F" "    800" " 5"  "M"
 [6,] "Jack"   "F" "1500000" " 6"  "B"
 [7,] "Tom"    "M" "   1200" " 7"  "B"
 [8,] "Mary"   "F" "   1400" " 8"  "B"
 [9,] "Joe"    "M" "    900" " 9"  "M"
[10,] "Bonnie" "F" "   1100" "10"  "M"
> f[6,2]<-'M'
> f
      Person   Sex RMB       Order Edu
 [1,] "Alan"   "M" "   1000" " 1"  "D"
 [2,] "Irina"  "F" "   2000" " 2"  "D"
 [3,] "Mark"   "M" "    500" " 4"  "D"
 [4,] "Lisa"   "F" "   1500" " 3"  "D"
 [5,] "Swan"   "F" "    800" " 5"  "M"
 [6,] "Jack"   "M" "1500000" " 6"  "B"
 [7,] "Tom"    "M" "   1200" " 7"  "B"
 [8,] "Mary"   "F" "   1400" " 8"  "B"
```

图 2.9 用 R 创建数据框

同样，我们也用 Python 进行数据型塑。不过，因为所需要处理的是缺失值，而不是“变性”问题，索性我们就只要填写 NA 的数值，即可。代码如下：

```
f = f.fillna('M')
# 函数.fillna 是填满（fill）缺失值（Na）的方法。
```

没错！这次 Python 只用一行代码。应该出现：

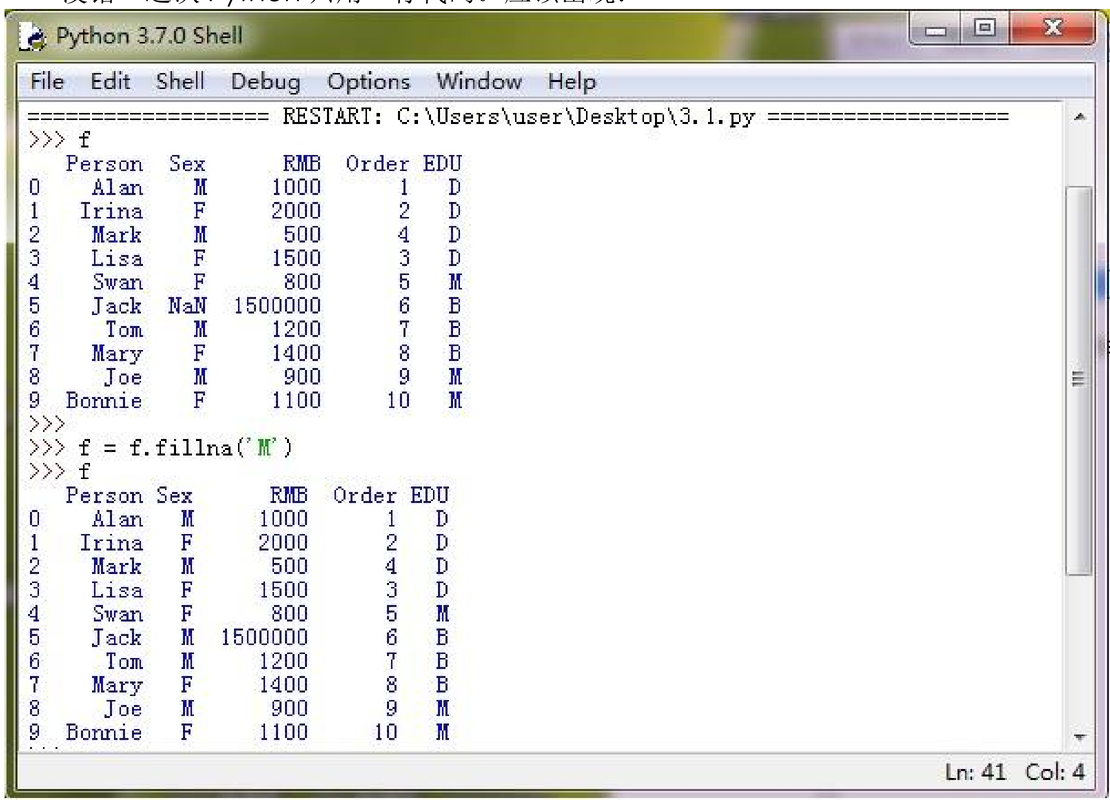


图 2.10 用 Python 填满缺失值

二、行列选择

接着，我们想要从 R 矩阵里，抽取部分数据，形成新的小数据集。

这里可以：用行抽取、用列抽取、行列抽取...等。很多种形式，我们也可以根据条件式编写筛选条件，这些操作与数据库相同。在此，我们姑且列举三种。

```
f1<-f[1:2,]
f1
# 选择第一行、第二行，共两行。
f2<-f[,c(2,5)]
f2
# 选择第二列和第五列，共两列。
f3<-f[1:2,c(2,5)]
f3
# 选择两行两列所包围的范围内的数据
```

如图所示

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助

> f1<-f[1:2,]
> f1
     Person  Sex RMB      Order Edu
[1,] "Alan"  "M" "   1000" " 1"  "D"
[2,] "Irina" "F" "   2000" " 2"  "D"
> f2<-f[,c(2,5)]
> f2
      Sex Edu
 [1,] "M" "D"
 [2,] "F" "D"
 [3,] "M" "D"
 [4,] "F" "D"
 [5,] "F" "M"
 [6,] "M" "B"
 [7,] "M" "B"
 [8,] "F" "B"
 [9,] "M" "M"
[10,] "F" "M"
> f3<-f[1:2,c(2,5)]
> f3
     Sex Edu
[1,] "M" "D"
[2,] "F" "D"
```

图 2.11 用 R 抽取数据

接着，我们试试用 Python 的 Pandas 的数据框里，去抽取数据。同样，我们采取：用行抽取、用列抽取、行列抽取...等三种。

```
f1=f[0:2]
f
# 选择了 0 到 1 的两行，实际上就是 Alan 和 Irina 的那两行。
f2=f[['Sex','EDU']]
f2
# 选择了“Sex”和“EDU”的那两列。
f3=f.loc[0:2,['Person','Sex','EDU']]
f3
# 选择了 0 到 1 的两行，人物、性别、教育的三个列，这两行三列所包含的数据。
# 在 Python 3x 使用.loc 方法，但在 python 2x 需要 iloc 的索引方法。
```

如图所示

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>> f1=f[0:2]
>>> f1
  Person Sex   RMB  Order EDU
0   Alan   M  1000      1   D
1  Irina   F  2000      2   D
>>>
>>> f2=f[['Sex','EDU']]
>>> f2
  Sex EDU
0   M   D
1   F   D
2   M   D
3   F   D
4   F   M
5   M   B
6   M   B
7   F   B
8   M   M
9   F   M
>>>
>>> f3=f.loc[0:2,['Person','Sex','EDU']]
>>> f3
  Person Sex EDU
0   Alan   M   D
1  Irina   F   D
2   Mark   M   D
>>>
                                              Ln: 92  Col: 4
```

图 3.12 用 Python 抽取数据

第三节 数据合并

一、利用矩阵的方法

前面我们小范围内地试验了数据抽取，与之相反，这个章节需要测试数据合并。常用的方法有两种，一是就数据本身，二是依靠索引。

此处，我们借用 R 来进行对于数据本身的调整与合并，顺便复习一下第二章的部分内容。代码如下：

```
f4<-f[1:4,c(1,2,5)]
f5<-f[5:8,c(1,2,5)]
# 分别抽取 f 里的部分行列，形成 f4 和 f5 的两个对象。

f4<-as.data.frame(f4)
f5<-as.data.frame(f5)
# 此处，英文里的 as，有“作为....”的意思，所以我们把() 内的对象，作为数据框。

f4
f5
f6<-merge(f4,f5,by="Sex")
f6
# 英文 by 是“通过....”的意思，所以这里很好理解，通过“性别”这个变量，把 f4
和 f5 进行合并（Merge）。然而，实际上，这里用到了键（key）值(Value)关系的对。
```

因为在第二节第一部分里，我们把 f 从数据框变成矩阵形式（二维的列表），所以这里我们想要重新采取数据框操作时，又把它们变回数据框。然后在数据框里，进行合并。

应该出现：

```
> f4
  Person Sex Edu
1   Alan   M   D
2  Irina   F   D
3   Mark   M   D
4   Lisa   F   D
> f5
  Person Sex Edu
1   Swan   F   M
2   Jack   M   B
3    Tom   M   B
4   Mary   F   B
> f6<-merge(f4,f5,by="Sex")
> f6
  Sex Person.x Edu.x Person.y Edu.y
1   F    Irina     D     Swan     M
2   F    Irina     D     Mary     B
3   F     Lisa     D     Swan     M
4   F     Lisa     D     Mary     B
5   M     Alan     D     Jack     B
6   M     Alan     D      Tom     B
7   M     Mark     D     Jack     B
8   M     Mark     D      Tom     B
>
```

图 3.13 用 R 合并数据

以上，我们完成合并的工作。在 R 里面的内嵌函数中，还有 cbind 和 rbind 等，德文 binden 有捆绑、结合、联盟的含义，在数据操作上，也可以这么理解。

接下来，我们想用 Python 来做同样的事情，并且把 Merge 理解得更为透彻一些。

二、利用索引的方法

我们现在考虑一个问题，就是数据量一旦很大的情况下，每次都要调用所有数据，塞到小小的计算机上，那么内存或者中央处理器，负担压力很大。如果，我们对于数据建立一个索引，我们只要进行索引的操作，最后再去调取数据，那么计算压力（计算资源）就会大幅减轻。

如果考虑这个问题，那么，就能同时理解两件事情：

（一）在 python 里，特别是处理数值的 numpy 和处理数据库性质工作的 pandas 上，应该会是特别注意到索引的功能。

（二）键（key）值(Value)关系的对，在 python 里有专门的数据类型，已经定义好在 Python 内核里的字典（Dicitonary）就是专门处理海量数据的一种可靠做法。尽管，当初设计没有特别考虑到数据框和数据库操作的情况，但是基本原理相通。

代码如下：

```
f4 =f.loc[0:3,['Person','Sex','EDU']]
f4
# 我们利用.loc 选择了 4 行和 3 列。
f5 =f.loc[4:7,['Person','Sex','EDU']]
f5
f6 = pd.merge(f4,f5, on = 'Sex')
```

```
f6
#利用 pandas 库的 merge 方法，把两个对象 f4 和 f5 按照共同的一个键 Sex 给合并起来。
```

XXXXXX

应该出现：

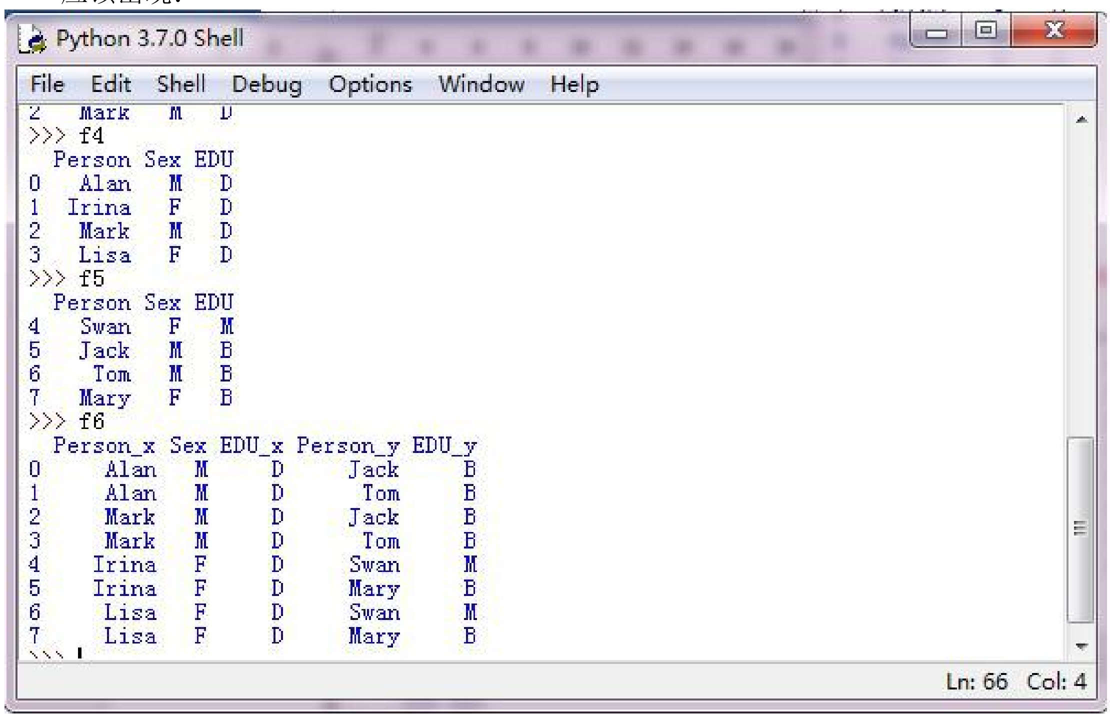

```
2   Mark   M   D
>>> f4
  Person Sex EDU
0   Alan   M   D
1  Irina   F   D
2   Mark   M   D
3   Lisa   F   D
>>> f5
  Person Sex EDU
4   Swan   F   M
5   Jack   M   B
6    Tom   M   B
7   Mary   F   B
>>> f6
  Person_x Sex EDU_x Person_y EDU_y
0     Alan   M     D     Jack     B
1     Alan   M     D      Tom     B
2     Mark   M     D     Jack     B
3     Mark   M     D      Tom     B
4    Irina   F     D     Swan     M
5    Irina   F     D     Mary     B
6     Lisa   F     D     Swan     M
7     Lisa   F     D     Mary     B
>>>
```

图 3.14 用 Python 合并数据

以上，我们比较图 3.13 和图 3.14 时，可以了解到所处理的数据合并的结果是一样的；比较第一部分和第二部分的代码时，可以了解到，它们的代码虽然很像，但是实际过程并不太相同。由此，可以做一番简短思考后，该换用其他代码试试。

本章小结

习　　题

一、单选题

1、如果我们遇到异常值，应该首先做什么？（___）

（A）删除它（B）忽视它（C）利用平均数代替它（D）了解异常值的来源

2、请问 NA 符号，代表什么意思？（___）

（A）缺失值（B）空值（C）异常值（D）注释

3、我们可以使用 R 的 reorder()函数，根据数据数值改变因子的顺序。

iss <- InsectSprays

```
iss$spray
iss$spray <- reorder(iss$spray, iss$count, FUN=mean)
iss$spray
```
请问 iss$spray, iss$count, FUN=mean 各自代表了 reorder()函数的：（___）
（B）因子、排序依据的数据、汇总数据的函数
（C）排序依据的数据、因子、汇总数据的函数
（D）排序依据的数据、汇总数据的函数、因子
（E）汇总数据的函数、排序依据的数据、因子

4、在 Python 里【列表】与【元组】的区别在于（___）
（A）列表的数据项之间以逗号分隔
（B）列表可以嵌套定义
（C）列表可以是不同的数据类型
（D）创建列表需要使用方括号

二、 多选题

1、把连续变量转换为分类变量，操作顺序是（___）
（1）使用 labels 参数，改变因子水平向量的因子名称
（2）使用 include.lowest 参数，设为 TRUE，生成区间
（3）使用 right 参数，设为 FALSE，把区间定义为左封右开
（4）使用 cut()函数；

三．问答题

1、数据型塑和数据造假，有何区别？我们如何保证或者检验究竟是造假还是研究？
4、请说明什么是因子水平？

四．课后作业

如果您完成第二章的课后作业，那么第三章的作业就是增加一些分析。
根据第三章的内容，在原有的.csv 文件里，加入新的解读的内容。

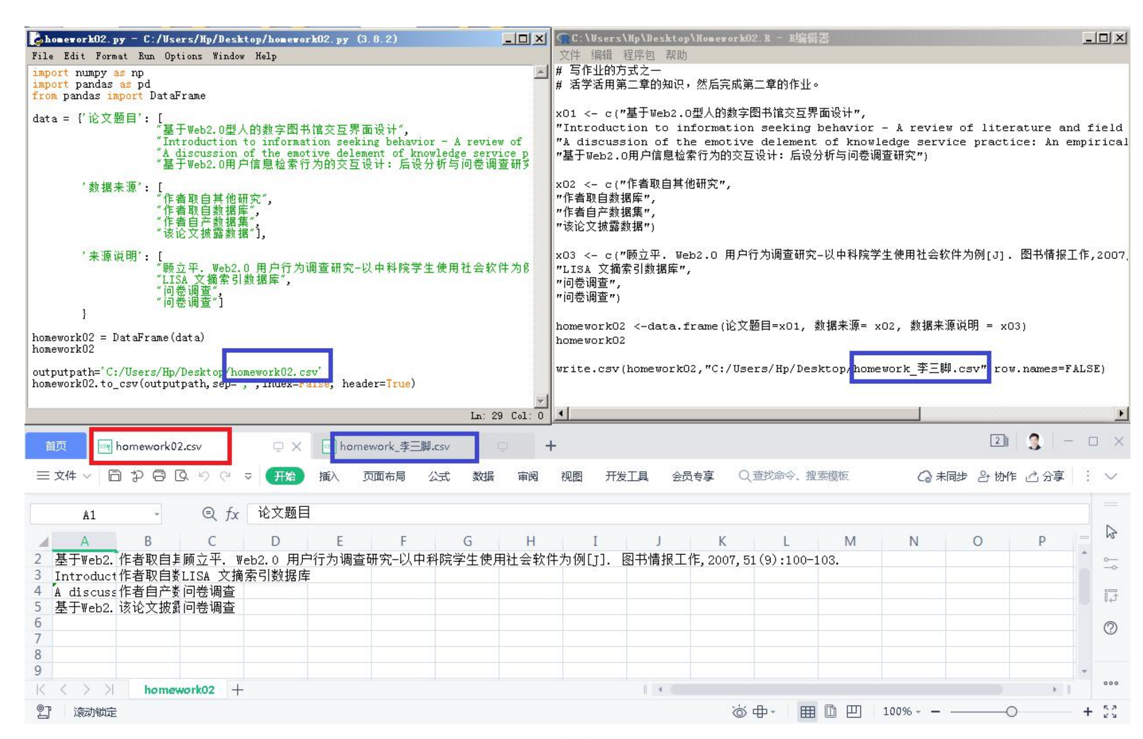


图 3.xx 第二章作业范例

1.这周作业：解读什么呢？

（1）原始数据到研究数据的数据版本的格式是否调整?

1-1.原始数据的格式：csv、txt...等。

1-2.研究数据的格式：csv、txt...等。

（2）采用什么软件工具？

1-3.数据采集：没有、网页抓爬、录音设备、观测望远镜...等

1-4.数据处理：没有、R、Python、JAVA、ANMAS、MatLab...等

1-5.数据型塑：没有、R、Python、JAVA...等

1-6.数据分析（下一章的内容的预习）：没有、R、Python、JAVA...等

（3）数据集是否经过调整？

1-7.衍生数据：统计报表、数据抽取/合并、平均/删除（缺失值、异常值）...等。

1-8.变量类型：字符变成数字、数字变成字符、增加新的变量、增加新的因子...等。

1-9.行列转置：增、删、修、剪行列。

2.这周作业：怎么做呢？

参考图 3.xx 第二章作业范例，如用 R 进行整理。

如果你习惯读完一篇论文，就做一次记录，做完再做第二篇，那么可以考虑以下方式。

```
# 活学活用第二章的 R 知识、第三章的数据科学的知识，然后完成第三章的作业。

x01 <- c(" "," "," "," "," "," "," "," "," "," ")
```

```
x02 <- c(" "," "," "," "," "," "," "," "," "," ")
x03 <- c(" "," "," "," "," "," "," "," "," "," ")
x04 <- c(" "," "," "," "," "," "," "," "," "," ")
x05 <- c(" "," "," "," "," "," "," "," "," "," ")
x06 <- c(" "," "," "," "," "," "," "," "," "," ")
x07 <- c(" "," "," "," "," "," "," "," "," "," ")
x08 <- c(" "," "," "," "," "," "," "," "," "," ")
x09 <- c(" "," "," "," "," "," "," "," "," "," ")
x10 <- c(" "," "," "," "," "," "," "," "," "," ")
x11 <- c(" "," "," "," "," "," "," "," "," "," ")
x12 <- c(" "," "," "," "," "," "," "," "," "," ")

homework03 <-data.frame(论文题目= x01, 数据来源= x02, 来源说明 = x03
                原始格式= x04, 研究格式= x05, 数据采集 = x06
                数据处理= x07, 数据型塑= x08, 数据分析 = x09
                衍生数据= x10, 变量类型= x11, 行列转置 = x12)
homework03
write.csv(homework02,"C:/Users/Hp/Desktop/homework03.csv",row.names=FALSE)
```

参考图 3.xx 第二章作业范例，如用 Python 进行整理。

如果习惯根据一个问题，一次看十篇论文，看完再看第二个问题，那么可以考虑：

```
import numpy as np
import pandas as pd
from pandas import DataFrame
data = {'论文题目': [" ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " "],
    '数据来源': [" ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " ",
                    " "],
    # ....(以此类推)      }

homework03 = DataFrame(data)
homework03
outputpath='C:/Users/Hp/Desktop/homework03.csv'
```

```
homework03.to_csv(outputpath,sep=',',index=False, header=True)
```

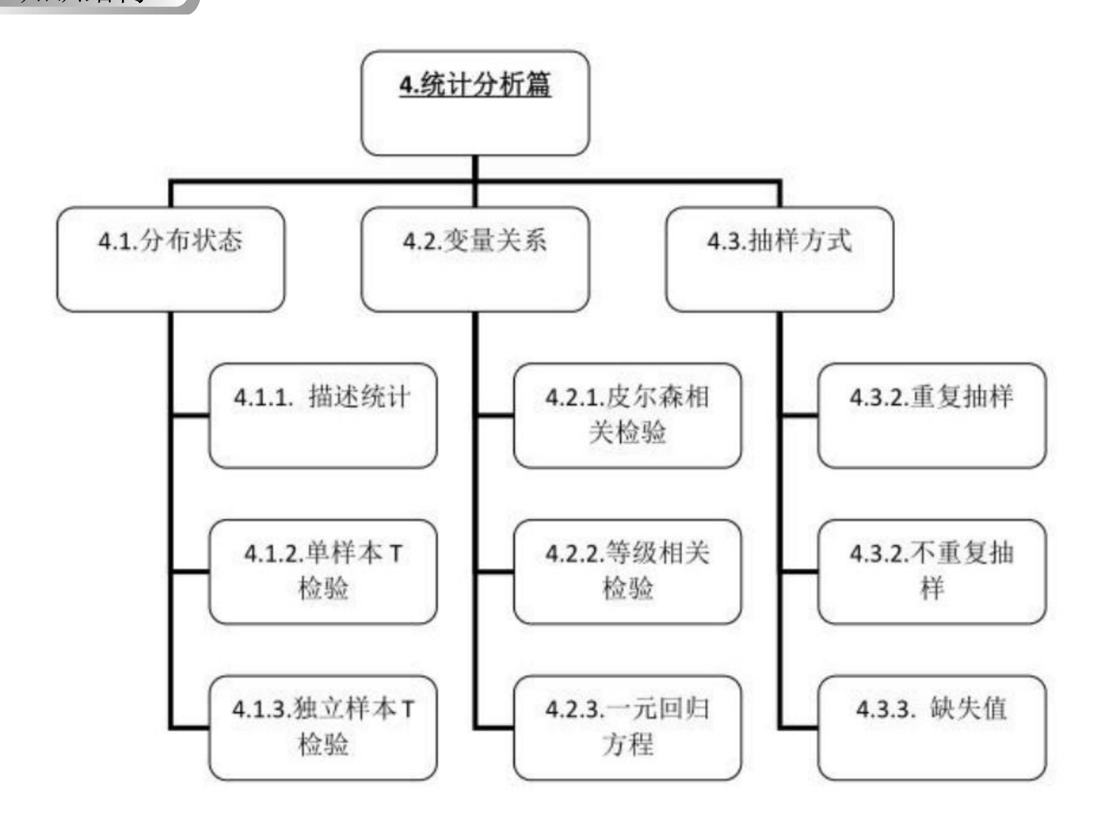
3. 我们需要既用 R 又用 Python 来做两次作业么？
当然不用。我们只需要选择其中一种来完成这次作业就可。
在第二章，我们为了了解 R 和 Python 两种互换方式，所以建议两种代码自己写。
如同上面的两个代码示例，您可以选择适合您的习惯的方式，择一即可。

第四章　统计分析篇

学习目标

掌握正态分布检验、单样本检验、组间检验等；
理解假设检验和统计推论；
掌握处理相关分析、变量等；
理解抽样、概率、置信区间等；
掌握处理缺失值的技巧。

知识结构



导入案例

本章节我们修订一下之前的数据框，修正了缺失值、异常值、空值，并且把行增加为10 行，把表头（变量名称）都写在三个英文字母以内，这样便于整齐美观。

我们可以键入 R Console 后，一条条运行，即第二章第一节的方法；或者，写在.r 文件，接着一次运行，即第二章第五节的方法。我们也可写成.csv 或者.txt 的文件，在需要使用时，导入。

代码示例，如下：

```
x1 <- c("Alan","Irina","Mark","Lisa","Swan","Jack","Tom","Mary","Joe","Bonnie")
x2 <- c("M","F","M","F","F","M","M","F","M","F")
x3 <- c(1000,2000,500,1500,800,3000,1200,1400,900,1100)
x4 <- c(1,2,4,3,5,6,7,8,9,10)
x5 <- c("D","D","D","D","M","B","B","B","M","M")
x6 <- c(77,86,96,95,76,85,96,55,62,66)
x7 <- c(76,88,100,89,74,99,62,52,64,56)
x8 <- c(78,85,95,96,75,83,97,57,64,69)
x9 <- c(5,2,6,3,5,1,4,3,5,4)
x10 <- c(6,9,5,8,6,10,7,8,6,7)

f <-data.frame(NAM = x1, SEX = x2, RMB = x3, ORD = x4, EDU= x5,
CHI = x6, MAT = x7, ENG = x8, ANG = x9, HAP = x10)
f
```

上述代码，我们可以键入 R Console 后，一条条运行；或者，写在.r 文件，接着一次运行。

所示结果，如下：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> x4 <- c(1,2,4,3,5,6,7,8,9,10)
> x5 <- c("D","D","D","D","M","B","B","B","M","M")
> x6 <- c(77,86,96,95,76,85,96,55,62,66)
> x7 <- c(76,88,100,89,74,99,62,52,64,56)
> x8 <- c(78,85,95,96,75,83,97,57,64,69)
> x9 <- c(5,2,6,3,5,1,4,3,5,4)
> x10 <- c(6,9,5,8,6,10,7,8,6,7)
>
> f <-data.frame(NAM = x1, SEX = x2, RMB = x3, ORD = x4, EDU= x5,
+ CHI = x6, MAT = x7, ENG = x8, ANG = x9, HAP = x10)
> f
      NAM SEX  RMB ORD EDU CHI MAT ENG ANG HAP
1    Alan   M 1000   1   D  77  76  78   5   6
2   Irina   F 2000   2   D  86  88  85   2   9
3    Mark   M  500   4   D  96 100  95   6   5
4    Lisa   F 1500   3   D  95  89  96   3   8
5    Swan   F  800   5   M  76  74  75   5   6
6    Jack   M 3000   6   B  85  99  83   1  10
7     Tom   M 1200   7   B  96  62  97   4   7
8    Mary   F 1400   8   B  55  52  57   3   8
9     Joe   M  900   9   M  62  64  64   5   6
10 Bonnie   F 1100  10   M  66  56  69   4   7
>
> |
```

图 4.1 新的试验数据集（R）

同理，我们也可使用 Python 建立同样的模块.py 文件，或者，数据集.csv 文件。

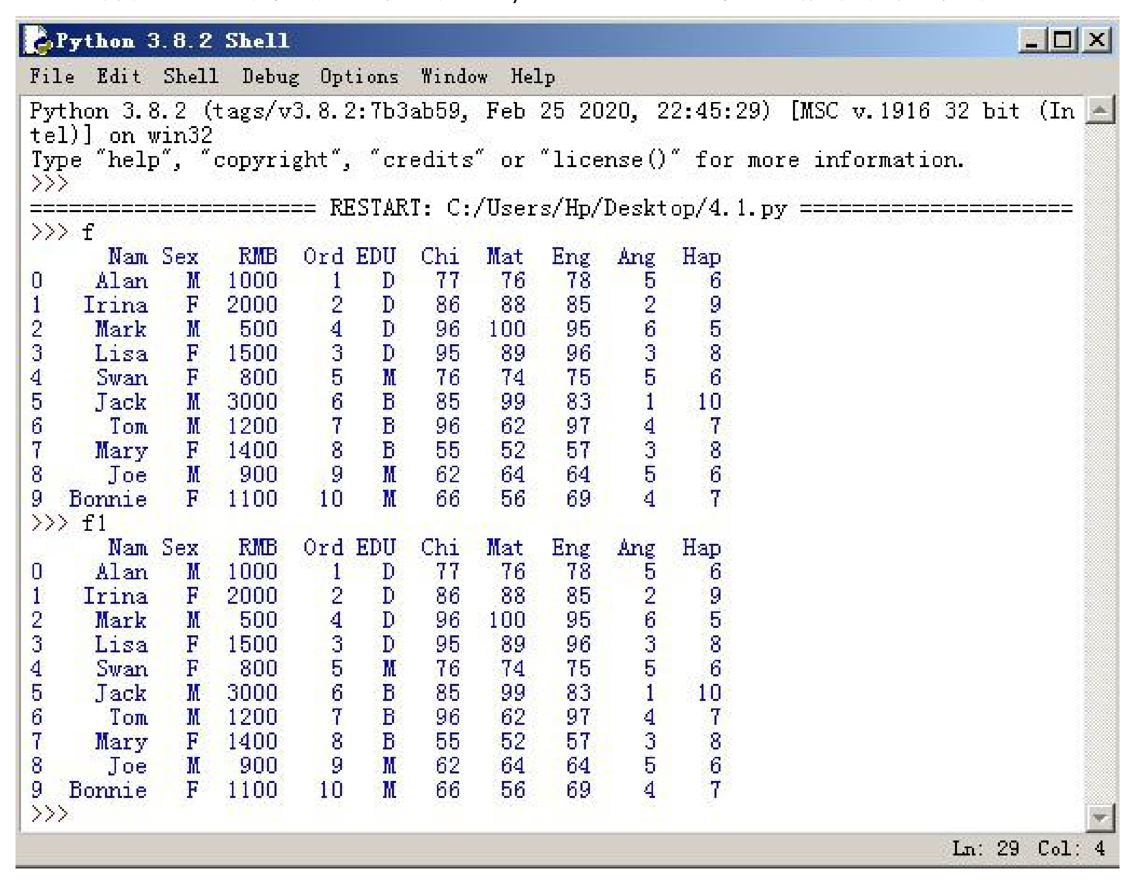
```
import numpy as np
import pandas as pd
from pandas import DataFrame
data = {'Nam': ["Alan","Irina","Mark","Lisa","Swan","Jack","Tom","Mary","Joe","Bonnie"],
```

```
        'Sex': ["M","F","M","F","F","M","M","F","M","F"],
        'RMB': [1000,2000,500,1500,800,3000,1200,1400,900,1100],
        'Ord': [1,2,4,3,5,6,7,8,9,10],
        'EDU': ["D","D","D","D","M","B","B","B","M","M"],
        'Chi': [77,86,96,95,76,85,96,55,62,66],
        'Mat': [76,88,100,89,74,99,62,52,64,56],
        'Eng': [78,85,95,96,75,83,97,57,64,69],
        'Ang': [5,2,6,3,5,1,4,3,5,4],
        'Hap': [6,9,5,8,6,10,7,8,6,7],
            }

f = DataFrame(data)

outputpath='C:/Users/Hp/Desktop/f7.csv'
f.to_csv(outputpath,sep=',',index=False, header=True)
f1 = pd.read_csv('C:/Users/Hp/Desktop/f7.csv')
```

我们把上述代码，写为.py 文件（例如，命名为 4.0.py）接着运行。在 Python Shell 里我们再键入 f 把对象调用出来，放在 Python Shell 的环境里。所示结果，如下：



```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Users/Hp/Desktop/4.1.py =====================
>>> f
      Nam Sex   RMB  Ord EDU  Chi  Mat  Eng  Ang  Hap
0    Alan   M  1000    1   D   77   76   78    5    6
1   Irina   F  2000    2   D   86   88   85    2    9
2    Mark   M   500    4   D   96  100   95    6    5
3    Lisa   F  1500    3   D   95   89   96    3    8
4    Swan   F   800    5   M   76   74   75    5    6
5    Jack   M  3000    6   B   85   99   83    1   10
6     Tom   M  1200    7   B   96   62   97    4    7
7    Mary   F  1400    8   B   55   52   57    3    8
8     Joe   M   900    9   M   62   64   64    5    6
9  Bonnie   F  1100   10   M   66   56   69    4    7
>>> f1
      Nam Sex   RMB  Ord EDU  Chi  Mat  Eng  Ang  Hap
0    Alan   M  1000    1   D   77   76   78    5    6
1   Irina   F  2000    2   D   86   88   85    2    9
2    Mark   M   500    4   D   96  100   95    6    5
3    Lisa   F  1500    3   D   95   89   96    3    8
4    Swan   F   800    5   M   76   74   75    5    6
5    Jack   M  3000    6   B   85   99   83    1   10
6     Tom   M  1200    7   B   96   62   97    4    7
7    Mary   F  1400    8   B   55   52   57    3    8
8     Joe   M   900    9   M   62   64   64    5    6
9  Bonnie   F  1100   10   M   66   56   69    4    7
>>>
Ln: 29  Col: 4
```

图 4.2 新的试验数据集（Python）

这些试验数据集，不能拿来作为科学研究，只是为了方便我们了解和操作 R 与 Python 而已，为了便于讨论，我们姑且给于各个行列的含义如下：

NAM 姓名：定类变量，每个数值都不相同。

SEX 性别：定类变量，包括两种数值：男 M 和女 F

RMB 收入：定类变量

ORD 次序：系统里是定类变量，实际上无意义，不是变量。

EDU 教育：定类变量，包括三种数值：博士 D、硕士 M 和学士 B。
CHI 语文: 定比变量，数值范围从 0 到 100 但是实际上是 60 到 100 之间。
MAT 数学：定比变量，数值范围从 0 到 100 但是实际上是 60 到 100 之间。
ENG 英文：定比变量，数值范围从 0 到 100 但是实际上是 60 到 100 之间。
ANG 生气：定距变量，数值范围从 1 到 10
HAP 快乐：定序变量，数值范围从 1 到 10.

表 4.1. 变量类型

计算机认为：	领域知识认为：	统计认为：
字符串	定类尺度（nominal scale）	不具备任何数学特性。例如，标记、类别。
数字	定序尺度（ordinal scale）	具有逻辑顺序，如强度、程度或等级。
	定距尺度（interval scale）	事物之间的等级，具有距离和数量差别，数字可相加减。
	定比尺度（ratio scale）	事物之间的比例、倍数关系，可进行加减乘除运算。

除此之外，我们人为制造的这组数据集，为了日后便于各种练习操作，还有隐藏预设了如下的三种情况：

语文、数学、英文，将来作为 IQ 的数据，三者是正比关系。
生气和快乐，将来作为 EQ 的数据，生气和快乐是反比关系。
收入和快乐，其变量类型不同，但是呈现正比关系。

当我们准备好试验数据集之后，我们可以开始进行“依据变量的不同”适合采取哪种统计分析的探索研究的角度，简短扼要地介绍统计分析的基础内容。

第一节 分布状态

一、描述统计

拿到一笔数据集之后，如果经过第二章和第三章的数据处理、数据规整，现在我们开始进行统计分析，首先就是描述统计。在 R 里面，如下所示。

```
summary(f)
# 数据框的描述统计

summary(f[,7])
summary(x7)
# 列的描述统计
```

能够很直观地看到结果：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
9      Joe    M  900   9    M  62  64  64   5   6
10 Bonnie    F 1100  10    M  66  56  69   4   7
>
> summary(f)
       NAM      SEX         RMB              ORD          EDU         CHI
 Alan    :1    F:5   Min.   : 500   Min.   : 1.00   B:3   Min.   :55.00
 Bonnie  :1    M:5   1st Qu.: 925   1st Qu.: 3.25   D:4   1st Qu.:68.50
 Irina   :1          Median :1150   Median : 5.50   M:3   Median :81.00
 Jack    :1          Mean   :1340   Mean   : 5.50         Mean   :79.40
 Joe     :1          3rd Qu.:1475   3rd Qu.: 7.75         3rd Qu.:92.75
 Lisa    :1          Max.   :3000   Max.   :10.00         Max.   :96.00
 (Other):4
       MAT               ENG             ANG             HAP
 Min.   : 52.00   Min.   :57.0   Min.   :1.0   Min.   : 5.0
 1st Qu.: 62.50   1st Qu.:70.5   1st Qu.:3.0   1st Qu.: 6.0
 Median : 75.00   Median :80.5   Median :4.0   Median : 7.0
 Mean   : 76.00   Mean   :79.9   Mean   :3.8   Mean   : 7.2
 3rd Qu.: 88.75   3rd Qu.:92.5   3rd Qu.:5.0   3rd Qu.: 8.0
 Max.   :100.00   Max.   :97.0   Max.   :6.0   Max.   :10.0

> summary(x7)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  52.00   62.50   75.00   76.00   88.75  100.00
>
```

图 4.3 描述统计（R）

当然，我们最重要的是理解含义，就是读懂统计报表，而不是记忆代码。在图 4.3.的报表里的各个含义，如表 4.2 所示。

表 4.2. 描述统计的报表（R）

报表符号	中文名词	含义
Min	最小值	这一列是否可以删除
1st Qu.	上四分位数	
Median	中位数	

Mean	均值	
3rd Qu.	下四分位数	
Max	最大值	

同理，我们也可以在 Python 里进行同样工作

```
f.descrube()
# 对 f 数据框进行描述统计

f3 = f7["Mat"]
# 单独抽出一列

f3.descibe()
# 对 f 数据框的 Mat 列进行描述统计
```

能够很直观地看到相似结果：

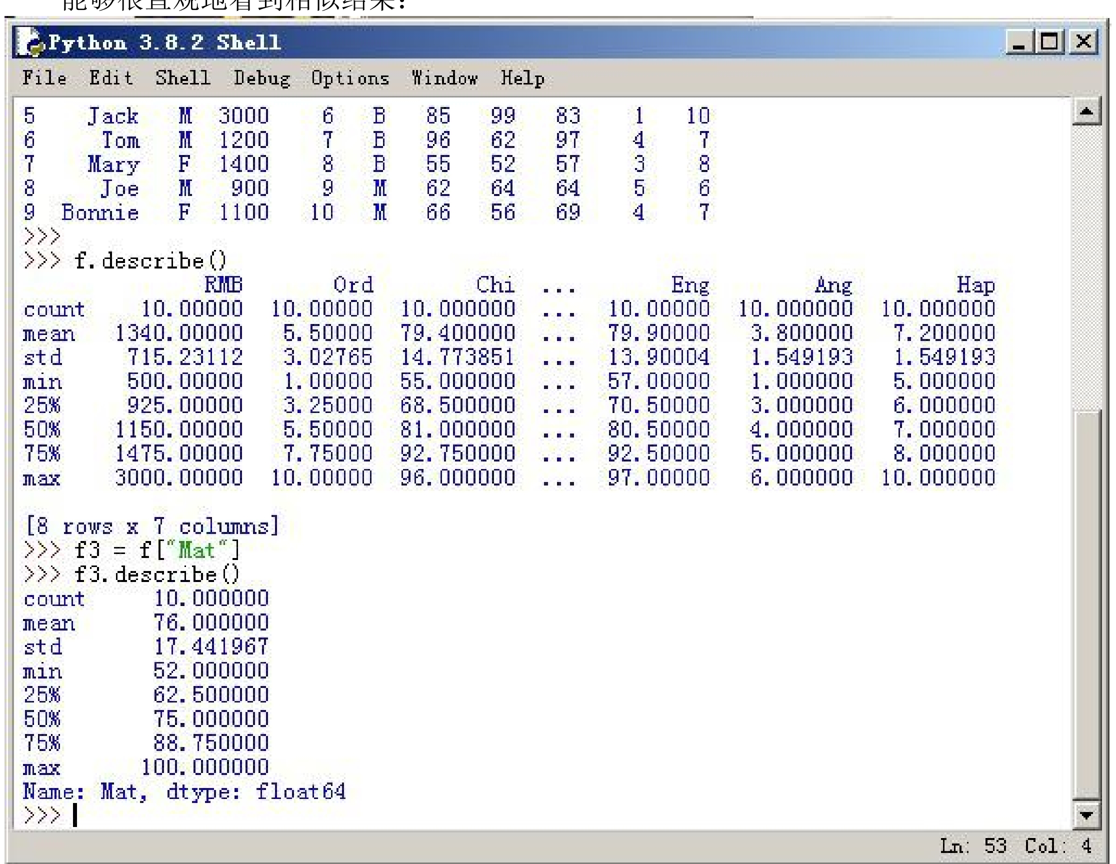


图 4.4 描述统计（Python）

我们最重要的是理解含义，如表 4.3 所示。

表 4.3. 描述统计的报表（Python）

报表符号	中文名词	含义
count	计数	
mean	平均数	
std	标准差	
min	最小值	
25%	25%分位数	

50%	50%分位数	
75%	75%分位数	
max	最大值	

就统计描述而言，还有很多指标。这里用以观察某个样本的分布情况（即，数据框的某列，作为定量变量，作为定比变量）的均值，或者某些变量的频数统计。

二、单样本 T 检验

单样本 t 检验，适用于某个变量的平均值与已知标准（或理论/假设）平均值进行比较(μ）。这个所谓的“已知标准”应当来自学科领域的理论、前人研究成果，或者先前研究中确定的值。

使用单样本 t 检验的前提是，数据呈现正态分布。换言之，需要进行正态性检验(shapiro-Wilk 检验)，代码如下：

```
shapiro.test(f[,7])
# 正态性检验（shapiro-Wilk 检验）
# 此处，统计报表显示，weight 符合正态性分布，可以满足单样本 t 检验的假设条件（P>0.05）。
```

假设检验的典型问题，包括：

（一）均值（m）是否等于理论平均值（μ）？

（二）均值（m）是否小于理论平均值（μ）？

（三）均值（m）是否大于理论平均值（μ）？

在进行假设检验时，首先需要定义相应的无效假设（H0），定义如下：

H0：m = μ

H0：m≤ μ

H0：m≥ μ

据此，相应的备择假设（H1）如下：

H1：m ≠ μ （不同）

H1：m> m（大于）

H1：m< μ（小于）

其中，假设一称为双向检验；假设二、三，称为单向检验。

现在，我们进行单样本 t 检验，代码如下：

```
t.test(f[,7], mu = 80, alternative = "less")
t.test(f[,7], mu = 60, alternative = "greater")
# 如果要检验 Mat 的均值是否小于 80（单向测试），用 less，反之，设定 greater
```

所试验的结果，如图所示：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助

> summary(x7)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  52.00   62.50   75.00   76.00   88.75  100.00
> shapiro.test(f[,7])

        Shapiro-Wilk normality test

data:  f[, 7]
W = 0.93138, p-value = 0.4616

> t.test(f[,7], mu = 80, alternative = "less")

        One Sample t-test

data:  f[, 7]
t = -0.72521, df = 9, p-value = 0.2434
alternative hypothesis: true mean is less than 80
95 percent confidence interval:
     -Inf 86.11078
sample estimates:
mean of x
       76
```

图 4.5 单样本 T 检验（R）

检验的 p 值为 0.2434，大于显着性水平 alpha = 0.05。

我们可以得出结论，Mat 的均值与 80 显着不同，p 值= 8.6e-05。

同理，我们使用 Python 进行正态性检验（shapiro-Wilk 检验），通过检验，接着进行单样本 T 检验。

```
import numpy as np
from scipy.stats import kstest
# 导入 sciy 库的 stats（统计）模块的 kstest 方法。

f4 = kstest(f["Mat"], 'norm')
# 输出结果中第一个为统计数，第二个为 P 值

from scipy import stats
# 导入 sciy 库的 stats（统计）模块。

f5 = stats.ttest_1samp(f["Mat"],80, axis=0)
f6 = stats.ttest_1samp(f["Mat"],60, axis=0)
# 单样本 T 检验，第一个位置是 Mat 列，第二个位置是给定均值。
```

应该出现：

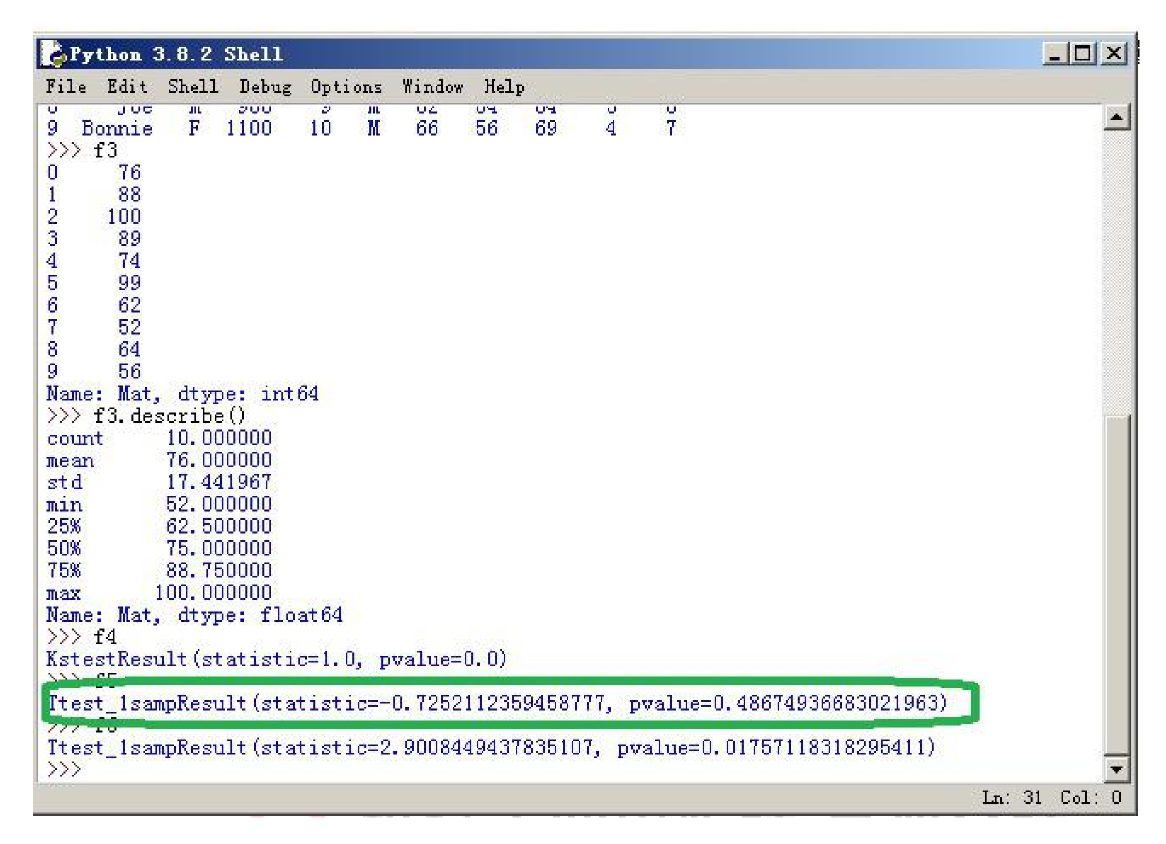


图 4.6 单样本 T 检验（Python）

比较图 4.5 与图 4.6 发现统计报表的结果一致，因此用 R 和 Python 均能作出单样本 T 检验。但是重点不在于此，而是需要对于报表进行解释。

以下，举例说明，可以如何解释图 4.5 与图 4.6 的统计报表。请注意，科学研究应该是先有研究问题、文献综述（所以知道“已知数值= 80”）、研究设计（假设检验）后，才是进行分析和解释，而不是像我们现在这样，利用试验数据集进行分析之后，才做事后解释的。

分析结果，如下：

已知某学校某班级的数学平均分数大于 80 分才是达标，现在随机抽取 10 位同学成绩（我们刚刚做的 10 笔数据的 Mat 列）

给定均值为 80 分（达标），结果显示 statistic=-0.7252112359458777（小于 0）这说明样本均值小于指定均值 80 了。

注意：这不表示该班的数学平均成绩低于 80 分。我们应看 pvalue: 0.48674936683021963 的情况。

说明：样本有>0.05 的概率认为数学平均成绩为 80。同理，不能拒绝 Mat 均值>80 的假设。

结论：我们接受该班数学成绩是达标的。

那么，请您试着解释一下，某校 60 分及格，那么现在这个班级是否及格？

三、两独立样本 T 检验

两独立样本 t 检验用于比较两个独立的组的均值是否存在差异。例如，试验数据集里，有 100 人，包括 50 名女性和 50 名男性；我们想知道女性的数学成绩和男性的数学成绩，是否不受它们各自性别的影响，那么，把男生视为独立样本，女生视为独立样本，它们的数学成绩又都具有正态分布的情况下，采用两独立样本 T 检验的方式，求证。

不过，在此，我们考虑试验数据集的两个变量的均值，中文和英文成绩，把它们视为两组不同的、独立的，不存在关系的独立样本，而且英文成绩和数学成绩都有正态分布（我们设计试验数据集的时候，刻意写成有分布的情况）的情况。

那么，在 R 里的代码，如下：

```
shapiro.test(f[,6])
shapiro.test(f[,8])
t.test(f[,6],f[,8], alternative = "two.sided", var.equal = FALSE)
```

应该出现：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助

> shapiro.test(f[,6])

        Shapiro-Wilk normality test

data:  f[, 6]
W = 0.91828, p-value = 0.3428

> shapiro.test(f[,8])

        Shapiro-Wilk normality test

data:  f[, 8]
W = 0.9422, p-value = 0.5777

> t.test(f[,6],f[,8], alternative = "two.sided", var.equal = FALSE)

        Welch Two Sample t-test

data:  f[, 6] and f[, 8]
t = -0.077947, df = 17.934, p-value = 0.9387
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -13.98027  12.98027
sample estimates:
mean of x mean of y
     79.4      79.9

>
```

图 4.7 两独立样本 T 检验（R）

同理，我们也用 Python 进行两独立样本 T 检验，比较图 4.7 和 4.8 的统计报表的结构是相同的。

```
stats.ttest_ind(f["Chi"],f["Eng"])
```

应该出现：

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
0      76
1      88
2     100
3      89
4      74
5      99
6      62
7      52
8      64
9      56
Name: Mat, dtype: int64
>>> f3.describe()
count     10.000000
mean      76.000000
std       17.441967
min       52.000000
25%       62.500000
50%       75.000000
75%       88.750000
max      100.000000
Name: Mat, dtype: float64
>>> f4 = kstest(f["Mat"], 'norm')
>>> f4
KstestResult(statistic=1.0, pvalue=0.0)
>>> f5
Ttest_1sampResult(statistic=-0.7252112359458777, pvalue=0.48674936683021963)
>>> f6
Ttest_1sampResult(statistic=2.9008449437835107, pvalue=0.01757118318295411)
>>> c.corr()
          value1    value2
value1  1.000000  0.656738
value2  0.656738  1.000000
>>> stats.ttest_ind(f["Chi"],f["Eng"])
Ttest_indResult(statistic=-0.077946534542498, pvalue=0.9387305278049435)
>>>
>>>
Ln: 65 Col: 4
```

图 4.8 两独立样本 T 检验（Python）

由 statistic=-0.077946534542498 小于 0 可以知道在样本统计上 Chi 的均值比 Eng 的小，事实上 Chi 的均值为 79.4，Eng 的均值为 79.9。另外 pvalue=0.9387305278049435 远大于 0.1，由此我们不能否认 Chi 均值与 Eng 均值存在明显差异，换言之，Chi 均值与 Eng 均值没有明显差别。

（再次注意，我们是刻意把 Chi 与 Eng 视为独立样本，事实上，它们是一个样本里的两个变量，我们为了便于分析，所以这么处理）

第二节 变量关系

一、皮尔森相关系数

皮尔逊（pearson）相关系数要求样本满足正态分布，两个变量之间的皮尔逊相关系数定义为两个变量之间的协方差和标准差的商，其值介于-1 与 1 之间。在 R 里的代码如下：

```
cor.test(f[,7],f[,8])
```

在 Python 里的代码如下，此时 Mat 和 Eng 不是数据框 f 里的两列，而是 Shell 里的两个对象。代码如下：

```
c = pd.DataFrame({'value1':f['Mat'],
                  'value2':f['Eng']})
c.corr()
```

二、斯皮尔曼等级相关系数

Sperman 秩相关系数是一种非参数统计量，其值与两组相关变量的具体值无关，而仅仅与其值之间的大小关系有关。皮尔森相关系数主要用于服从正太分布的连续变量，对于不是正态分布的变量，特别是定序变量，可采用 Sperman 秩相关系数。

在 R 里的代码如下：

```
cor.test(f[,9],f[,10],method = "spearman",)
```

在 Python 里的代码如下：

```
s = pd.DataFrame({'value1':f["Ang"],
                  'value2':f["Hap"]})
s.corr(method="spearman")
```

将上述 R 代码运行后应该出现：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
>
> cor.test(f[,7],f[,8])

        Pearson's product-moment correlation

data:  f[, 7] and f[, 8]
t = 2.4632, df = 8, p-value = 0.03912
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.04622627 0.91005642
sample estimates:
     cor
0.656738

> cor.test(f[,9],f[,10],method = "spearman",)

        Spearman's rank correlation rho

data:  f[, 9] and f[, 10]
S = 330, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
rho
 -1

Warning message:
In cor.test.default(f[, 9], f[, 10], method = "spearman", ) :
```

图 4.9 皮尔森和斯皮尔曼相关系数（R）

Python 代码运行后应该出现：

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
3    Lisa   F  1500   3  D  95  89  96  3   8
4    Swan   F   800   5  M  76  74  75  5   6
5    Jack   M  3000   6  B  85  99  83  1  10
6     Tom   M  1200   7  B  96  62  97  4   7
7    Mary   F  1400   8  B  55  52  57  3   8
8     Joe   M   900   9  M  62  64  64  5   6
9  Bonnie   F  1100  10  M  66  56  69  4   7
>>> f3
0     76
1     88
2    100
3     89
4     74
5     99
6     62
7     52
8     64
9     56
Name: Mat, dtype: int64
>>> f4
KstestResult(statistic=1.0, pvalue=0.0)
>>> f5
Ttest_1sampResult(statistic=-0.7252112359458777, pvalue=0.48674936683021963)
>>> f6
Ttest_1sampResult(statistic=2.9008449437835107, pvalue=0.01757118318295411)
>>> c.corr()
          value1    value2
value1  1.000000  0.656738
value2  0.656738  1.000000
>>> s.corr(method="spearman")
        value1  value2
value1     1.0    -1.0
value2    -1.0     1.0
>>> |
                                                        Ln: 55  Col: 4
```

图 4.10 皮尔森和斯皮尔曼相关系数（Python）

三、一元回归方程

因为我们通过前面第一部分确认 Mat（x7）和 Eng（x8）的关系，尽管是弱相关，但是存在相关性，因此我们可以试着就两者建立回归方程，假如理论上存在着这么一层更深入的关系的话。按照中学数学，一元方程的公式是：

Y = a X + b

我们需要知道的是 a 和 b 也就是斜率和拦截距。

R 的代码如下：

```
lm.reg<-lm(f[,8]~1+f[,7])
summary(lm.reg)
# 一元线性回归 Y = aX + b
# 从输出结果 p-值可以看出回归方程通过回归参数的检验与回归方程的检验。
# 由此得到回归方程 Y = 10.1557 + 0.8241 X
```

应该出现：

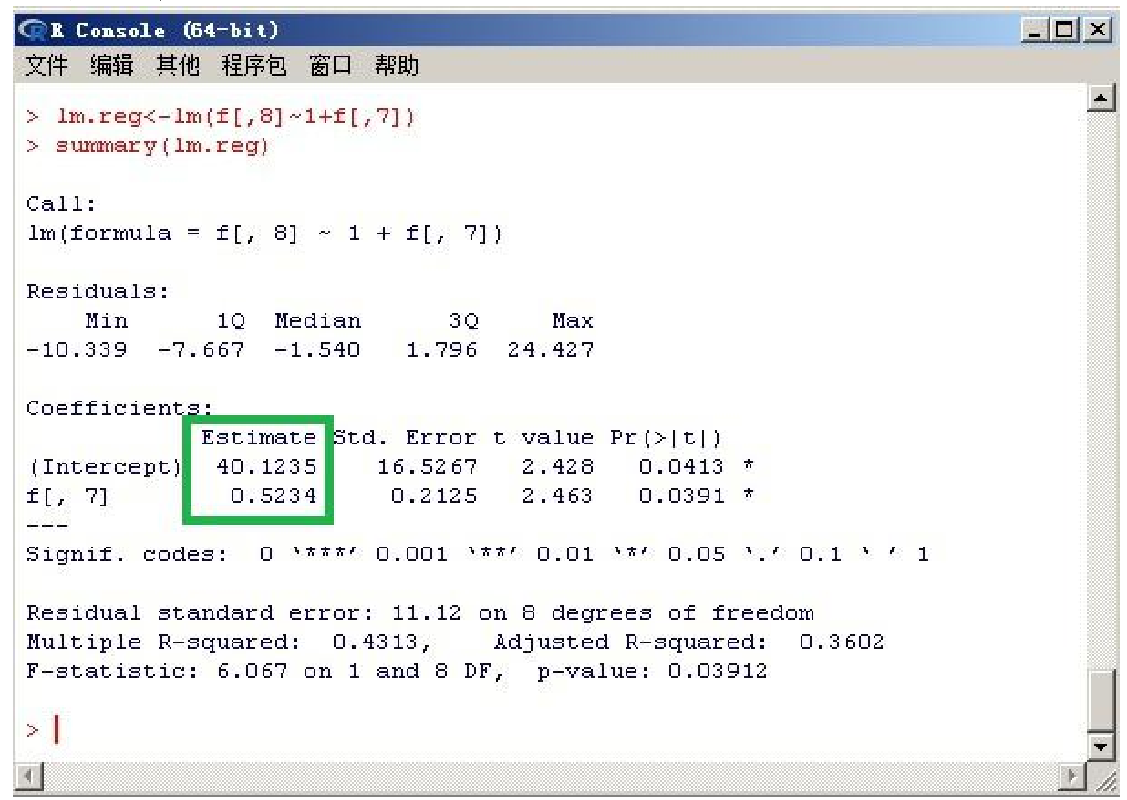


图 4.11 一元回归方程（R）

我们可以看到 Estimate(参数)所示的截距和斜率。从输出结果 p-值可以看出回归方程通过回归参数的检验与回归方程的检验，由此得到回归方程 Y= 40.1235 + 0.5234 X 即：a = 0.5234 和 b = 40.1235 的两组参数。在第七章，我们还会通过机器学习进行交叉验证。

由此，在 Python 里，可以做一番尝试。代码如下：

```
Mat = f['Mat'].values.reshape(-1, 1)
Eng = f['Eng'].values.reshape(-1, 1)
# 需要转换为介于-1 到 1 之间的数值
model = LinearRegression()
l= model.fit(Mat,Eng)
print("截距:",l.intercept_)
# 输出截距
print("斜率:",l.coef_)
# 输出斜率
```

应该出现：

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
7    Mary   F  1400    8   B   55   52   57    3    8
8     Joe   M   900    9   M   62   64   64    5    6
9  Bonnie   F  1100   10   M   66   56   69    4    7
>>> f3
0     76
1     88
2    100
3     89
4     74
5     99
6     62
7     52
8     64
9     56
Name: Mat, dtype: int64
>>> f4
KstestResult(statistic=1.0, pvalue=0.0)
>>> f5
Ttest_1sampResult(statistic=-0.7252112359458777, pvalue=0.48674936683021963)
>>> f6
Ttest_1sampResult(statistic=2.9008449437835107, pvalue=0.01757118318295411)
>>> c.corr()
          value1    value2
value1  1.000000  0.656738
value2  0.656738  1.000000
>>> s.corr(method="spearman")
        value1  value2
value1     1.0    -1.0
value2    -1.0     1.0
>>>
============================== RESTART: C:\Users\Hp\Desktop\4.1.py ============================
截距: [40.12352082]
斜率: [[0.52337473]]
>>> |
                                                                         Ln: 59  Col: 4
```

图 4.12 一元回归方程（Python）

比较图 4.11 和图 4.12 可知参数相同。其实，我们可以通过绘图，更为直观地看到 Mat 和 Eng 的相关关系。因此，在 R 里，我们可以接着继续：

```
windows <- par(mfrow=c(2, 2))
# 绘制 2 行 2 列的图像框（左上、左下、右上、右下）
plot(lm.reg)
par(windows)
# 对所得的回归方程中，误差项独立同正态分布的假设，进行检验。
# plot(lm.reg)实际上使用了四次 plot(x, y)而产生四个图形。
```

应该出现：

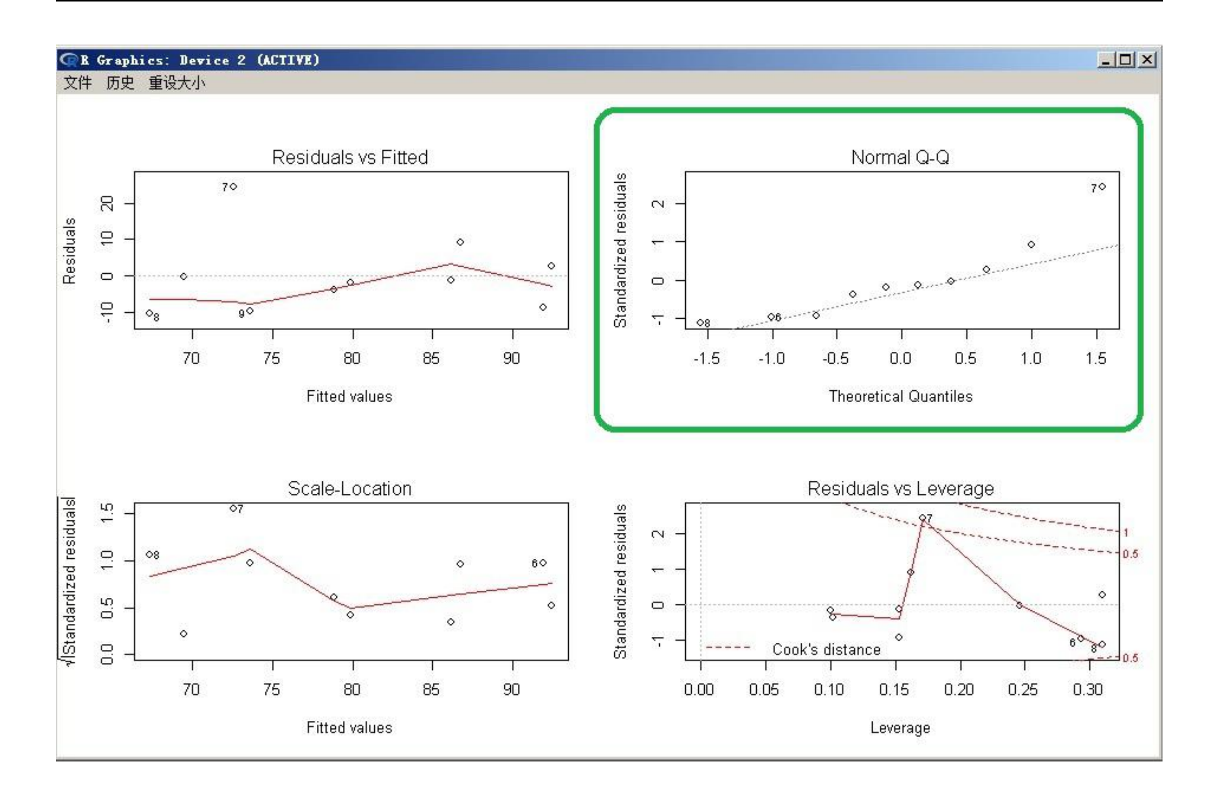


图 4.13 一元回归方程的示意图（R）

对于图 4.13 和上述代码，详细介绍，如下：

表 4.4 描述统计的报表（R）

位置	名称	中文	说明
左上	Residual vs ftted	拟合值ˆy 对残差的图	数据点基本均匀地分布在直线 y=0 的两侧，无明显趋势；
左下	Scale－Location	标准化残差(standardized residuals)的平方根的分布	最高点为残差最大值点；
右上	Normal QQ-plot	正态分布图	数据点分布趋于一条直线，说明残差是服从正态分布；
右下	Cook	距离图	对回归的影响点

同理，我们也可 Python 作图，如下代码：、

```
plt.plot(Mat,Eng,'b.')
plt.plot(Mat,model.predict(Mat),'r')
plt.show()
# 用 sklearn 和 malolib 实现的一元线性回归画图
```

应该出现：

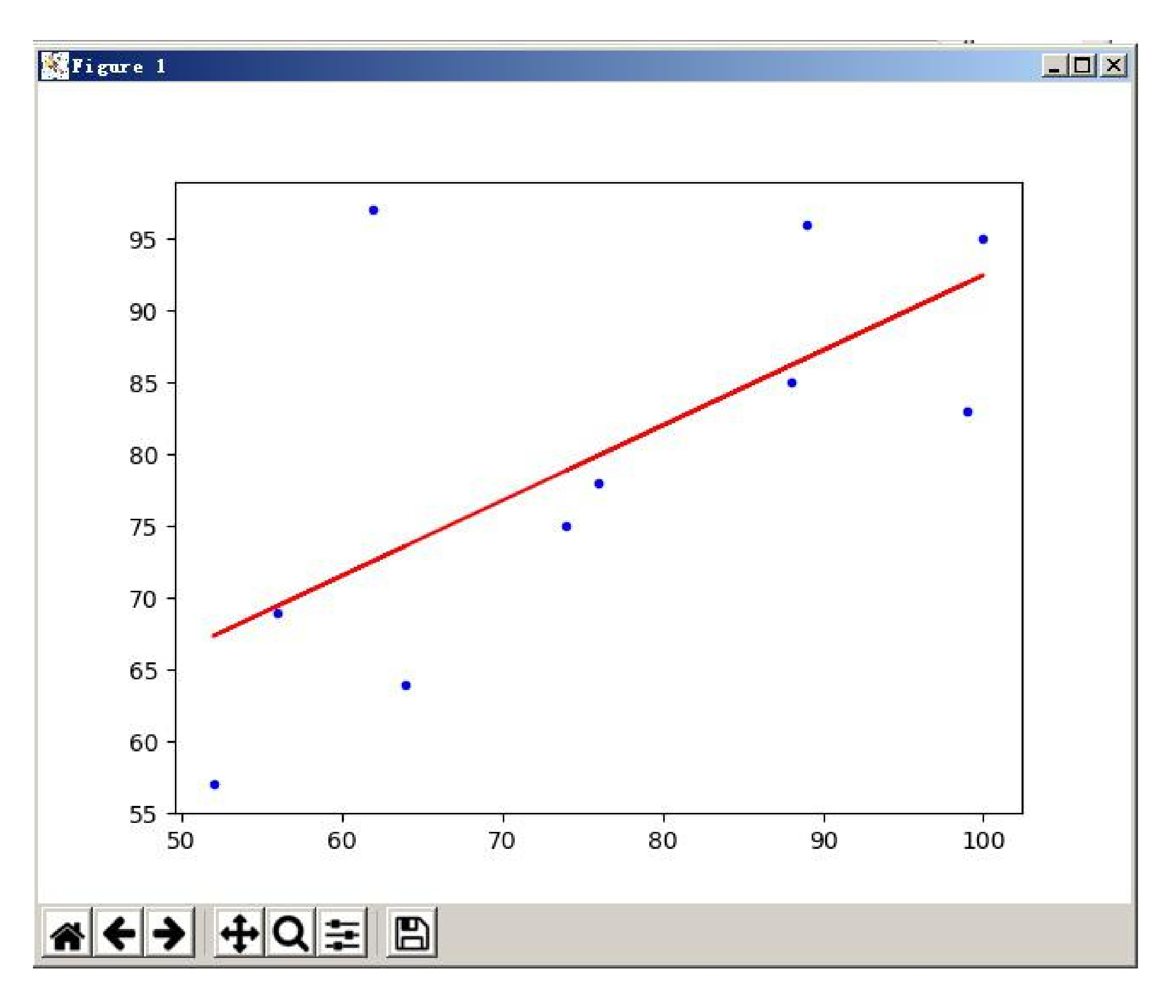


图 4.14 一元回归方程的示意图（Python）

关于图形，在下一章：数据可视化，会有所介绍。统计是收集数据、分析数据和由数据得出结论的一组概念、原则和方法。

当拥有来自一些变量(指标)的数据或记录,但缺乏模型来描述这些变量之间关系的情况下,可用统计方法建立模型。

建构模型后，利用统计方法可以确定数据是否足以支持某种论点；若干模型会作未来预测。统计图形，可以让学科领域专家快速掌握情况。

第三节 抽样方式

一、重复随机抽样

在 R 里的代码如下：

```
(x=1:100)
# 赋值 x 从 1 到 100 的数, 并且显示出来。
    sample(x,20)
# 从 x 抽出 20 个数字。
```

在 Python 里的代码如下

```
import random
N = range(100)
m = 20
a = random.sample(N, m)
print(a)
```

二、不重复随机抽样

在 R 里的代码如下：

```
x=sample(1:100,20)
y=sample(1:100,20)
z=sample(1:100,100)
# 从 1 到 100 中随机不放回地抽取 20、20、100 个值作为样本
x[1:10]
y[11:20]
z[91:100]
# 方括号中为 z 的下标，此处抽取 20 个的前 10 个、前 10 个、最后 10 个。
```

在 Python 里的代码如下：

```
import numpy as np
N = range(100)
a = np.random.choice(N, size=3, replace=False)
b = np.random.choice(N, size=6, replace=False)
c = np.random.choice(N, size=9, replace=False)
# a 抽样序列
# size 抽样数目
# replace 是否重复抽样。
```

```
print("抽三个:",a)
print("抽六个:",b)
print("抽九个:",c)
```

运行上述 R 代码后应该出现：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
>
>
> (x=1:100)
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
> sample(x,20)
 [1] 68 23 17 58 38 67 36 37 53 45 85 41  4 40 90 50 95 63 18 54
> z=sample(1:100,20)
> z[1:10]
 [1] 25 90 44 63 40 82 32 31 76 55
> x=sample(1:100,20)
> y=sample(1:100,20)
> z=sample(1:100,100)
> x[1:10]
 [1] 33 30 95 97 92 39 62 65 74  7
> y[11:20]
 [1] 16 39 44 30 70 21 49 83 57  3
> z[91:100]
 [1] 94 82 83 81 40  3  7 79 47 22
>
```

图 4.15 抽样（R）

运行上述 Python 代码后应该出现：

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
75%     1475.00000   7.75000  92.750000  ...  92.50000   5.000000   8.000000
max     3000.00000  10.00000  96.000000  ...  97.00000   6.000000  10.000000

[8 rows x 7 columns]
count      10.000000
mean       76.000000
std        17.441967
min        52.000000
25%        62.500000
50%        75.000000
75%        88.750000
max       100.000000
Name: Mat, dtype: float64
KstestResult(statistic=1.0, pvalue=0.0)
Ttest_1sampResult(statistic=-0.725211235945877, pvalue=0.48674936683021963)
Ttest_1sampResult(statistic=2.9008449437835107, pvalue=0.017571183182954 11)
Ttest_indResult(statistic=-0.0779465345424987, pvalue=0.9387305278049435)
          value1    value2
value1  1.000000  0.656738
value2  0.656738  1.000000
        value1  value2
value1     1.0    -1.0
value2    -1.0     1.0
截距: [40.12352082]
[36, 80, 47, 13, 17, 24, 77, 51, 4, 57, 82, 7, 61, 52, 54, 76, 60, 21, 45, 78]
抽三个: [ 4 69 83]
抽六个: [93 17 31 51 81 54]
抽九个: [96 18 61 10 68 24 83 63 25]
>>>
Ln: 63  Col: 4
```

图 4.16 抽样（Python）

三、随机森林

在 R 里的代码如下：

```
#读取一个有缺失值的 f7.csv 文件
f9 = read.csv("C:/Users/Hp/Desktop/f9.csv")
f9

complete.cases(f9)
#判断每行有没有缺失值
which(complete.cases(f9)==F)
#缺失值行号
```

表示如下：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> z=sample(1:100,100)
> x[1:10]
 [1] 22 52 97 11 73 16 55 82 17 66
> y[11:20]
 [1]  44  26  63  67 100  42  76  10  57  92
> z[91:100]
 [1] 45 40 93 70 77  6 20 91 26 29
> f9 = read.csv("C:/Users/Hp/Desktop/f9.csv")
> f9
      Nam Sex  RMB Ord EDU Chi Mat Eng Ang Hap
1    Alan   M 1000   1   D  77  76  78   5   6
2   Irina   F 2000   2   D  86  88  85   2   9
3    Mark   M  500   4   D  96  NA  ?5   6   5
4    Lisa   F 1500   3   D  95  89  96   3   8
5    Swan   F  800   5   M  76  74  75   5   6
6    Jack   M 3000   6   B  85  99  83   1  10
7     Tom   M 1200   7   B  96  62  97   4   7
8    Mary   F 1400   8   B  55  52  57   3   8
9     Joe   M  900   9   M  62  64  64   5   6
10 Bonnie   F 1100  10   M  66  56  69   4   7
> complete.cases(f9)
 [1]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> which(complete.cases(f9)==F)
[1] 3
```

图 4.17 读取和检查数据集（R）

在 R 里的代码如下：

```
# install.packages('missForest')# 安装 missForest 程序包
library(missForest)
# 启动 missForest 程序包
z=missForest(f9)
# 弥补缺失值
z
```

表示如下：

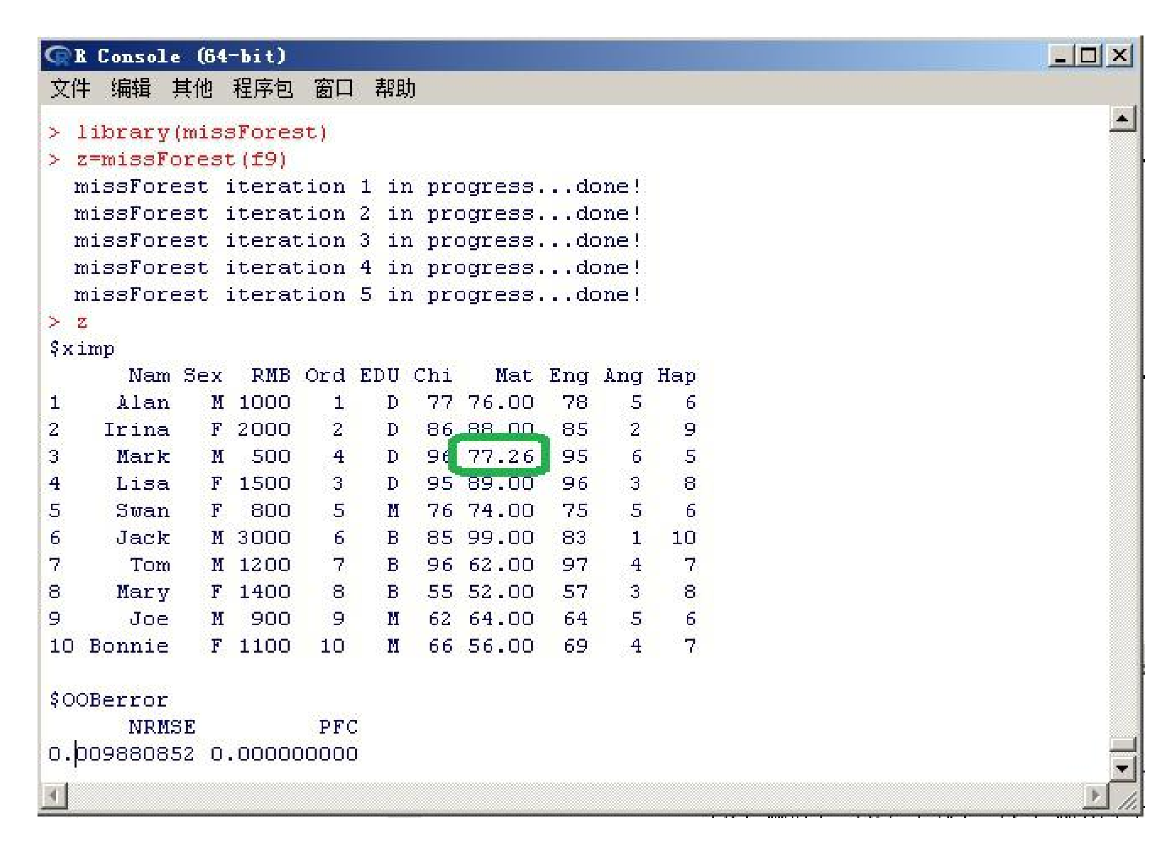


图 4.18 用随机森林填补缺失值（R）

在 R 里的代码如下：

```
f2=z$ximp
f2

f3<-na.omit(f9)
# 删去缺失值
f3
```

表示如下：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助

      Nam Sex  RMB Ord EDU Chi   Mat Eng Ang Hap
1    Alan   M 1000   1   D  77 76.00  78   5   6
2   Irina   F 2000   2   D  86 88.00  85   2   9
3    Mark   M  500   4   D  96 77.26  95   6   5
4    Lisa   F 1500   3   D  95 89.00  96   3   8
5    Swan   F  800   5   M  76 74.00  75   5   6
6    Jack   M 3000   6   B  85 99.00  83   1  10
7     Tom   M 1200   7   B  96 62.00  97   4   7
8    Mary   F 1400   8   B  55 52.00  57   3   8
9     Joe   M  900   9   M  62 64.00  64   5   6
10 Bonnie   F 1100  10   M  66 56.00  69   4   7
> f3<-na.omit(f9)
> f3
      Nam Sex  RMB Ord EDU Chi Mat Eng Ang Hap
1    Alan   M 1000   1   D  77  76  78   5   6
2   Irina   F 2000   2   D  86  88  85   2   9
4    Lisa   F 1500   3   D  95  89  96   3   8
5    Swan   F  800   5   M  76  74  75   5   6
6    Jack   M 3000   6   B  85  99  83   1  10
7     Tom   M 1200   7   B  96  62  97   4   7
8    Mary   F 1400   8   B  55  52  57   3   8
9     Joe   M  900   9   M  62  64  64   5   6
10 Bonnie   F 1100  10   M  66  56  69   4   7
> 
```

图 4.19 直接删除有缺失值的行（R）

本章小结

本章介绍了数据统计的知识，因为时间和篇幅有限，我们挑选了比较重要并且容易出错的应用统计学的部分，予以讲解介绍。本章主要讲述了以下内容：
（1）正态分布，涉及它的含义、假设检验，以及具体的正态分布检验、单样本检验、组间检验等三样，以 R 和 Python 的指令后的统计报表的解读，说明这些方法的使用场景、使用限制与条件，以及数据科学需要领域知识进行判断的理由。
（2）理解假设检验和统计推论，涉及科学方法上，量化计算以及有限度推论的原因，重点强调假设检验的逻辑；
（3）掌握处理相关分析、一元回归等，在变量之间是否可能存在相关关系的探索性研就的方式、流程原则等；
（4）理解抽样、概率、置信区间等；
（5）掌握处理缺失值的技巧，以及更为重要的是，如何看待缺失值。

习　　题

一、单选题

1、t 检验过程，是对两样本____差别的显著性进行检验。
（A）中位数
（B）均值
（C）标准差
（D）截率

2、单样本 t 检验：是用样本均数代表的未知总体均数和已知总体均数进行比较，来观察此组样本与总体的______。
（1）变异数
（2）斜率
（3）共同性
（4）差异性

二、多选题

1、拿到一组数据集，我们进行初步分析的顺序是（___）
（1）描述统计
（2）概率分布
（3）相关判断
（4）线性回归

2、拿到一组数据集，想要观察的变量之间的关系，正确顺序是（___）
（1）两个变量的关系是否显著？
（2）两个变量是否有关系？
（3）这个关系是不是因果关系
（4）这个关系是否带有普遍性？

三、问答题

1、请说明 Normal QQ-plot 图主要可以帮助我们做什么事情？使用时需要注意什么？

5、请说明独立样本 T 检验的正确用法？（注意，讲义所写不完全正确）

三、课后作业

如果您完成第二、第三章的课后作业，那么第四章的作业，分为两个步奏。

（1） 为 10 篇论文，设定编号。基于第三章的作业，这里增加一个编号。

（2） 为 10 篇论文，解读：【目的】、【方法】、【结论】各一句话（可以从论文摘要里截取）；这里，单独成为一个.csv 文件，包括四列：编号、目的、方法、结论。

（3） 作业（1）和（2）的编号，需要统一一致。

第五章　数据建模篇

学习目标

理解数据建模的概念、选型、流程等；
掌握社会网络的基本建模；
掌握网页排名的基本建模；
掌握信息推荐的基本建模；
掌握分布式计算的基本建模；
理解问题驱动算法优化的应用案例。

知识结构

- 5. 数据建模
  - 5.1. 建模
    - 5.1.1. 基本概念
    - 5.1.2. 误解误用
    - 5.1.3. 模型选择
    - 5.1.4. 实施要点
  - 5.2. 排名
    - 5.2.1. 社会网络
    - 5.2.2. 排名算法
    - 5.2.3. 算法实现
    - 5.2.4. 算法优化
  - 5.3. 推荐
    - 5.3.1. 推荐算法
    - 5.3.2. 分布计算
    - 5.3.3. 词频计算
    - 5.3.4. 相似推荐


导入案例

本章节开始，我们默认数据集来自实践场景，所不同的是我们在每小节采取的是小范围的范例；另外，我们默认 R 与 Python 的基本操作已经了然于胸，所以与第二、三、四、五章不同，此处代码示例不再复 R 与 Python 的对照，而是直接面向数据科学的解释和应用为主。

第一节 建模

一、基本概念

物理学家 Eugene Wigner 在 1960 年，发表著名论文《The Unreasonable Effectiveness of Mathematics in the Natural Sciences》，表明抽象数学概念的有效性远超出了他们创造者所能想象的。例如，非欧几何成为爱因斯坦理论的基础。

简言之，数学是优雅而且伟大的人类智力提升途径。

Google 研究人员 Alon Halevy, Peter Norvig, and Fernando Pereira 在 2009 年发表著名论文《The Unreasonable Effectiveness of Data》以语言翻译为例，表明有了足够的数据，数学模型的选择就不再重要。简单的模型和大量的数据远优于基于少量数据的更加雕琢的模型。

简言之，算法不够优雅，但是足够暴力，所以直接有效。

如您回忆第一章第一节内容的话，可以了解真实世界和数据世界，其实并不一定相同，而目前，我们在数据建模这章，还需要了解数据和模型的不同。

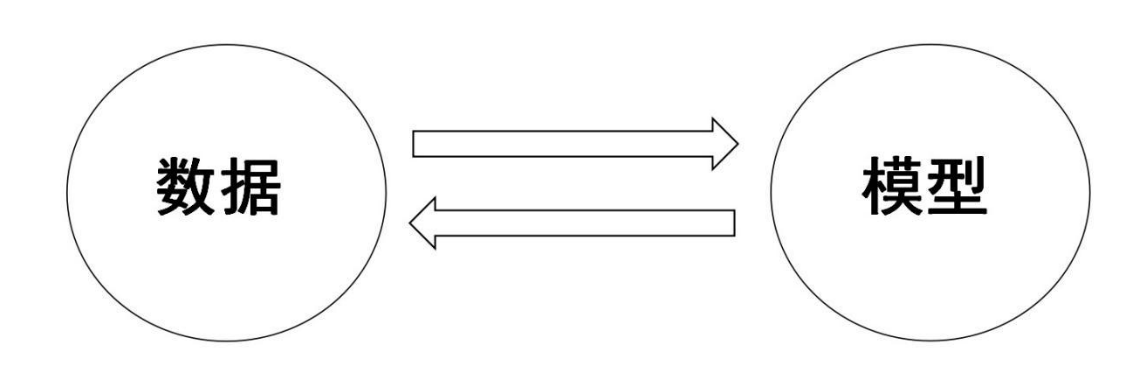


图 6.1 科学研究的形象化描述

什么是模型？模型是根据数据训练得来。方式为：

（一）探索性数据分析，看看数据的大致情况，人为主观确定出一个参数模型类型或者一个可能适用的算法。典型的参数模型，包括：线性回归模型等；典型的算法模型，包括：决策树、随机森林等。

（二）训练数据得到模型，是根据已知数据，进行机器学习和训练，求得参数模型的参数或者算法模型的程序。

那么，为了了解，我们姑且称为参数模型和算法模型，两种建模方式。

什么是参数模型？经典统计：假定正态分布的前提下，进行假设检验及点估计、线性回归、多元分析等，也就是目前多数教科书里面写的经典统计。其特点是重视数学，模型形式是根据经验及数学的可算性假定的可以写出公式的模型，模型参数由数据来估计。其判断优劣的方法，着重在：拟合优度检验、t 检验、F 检验，p 值、残差分析等。其误差的含义，是指：数据和模型的差距，以及参数解的精确性，但只能核对后者，因为只有一个训练集（自己对自己投票）没有测试集（这个部分内容在第七章介绍）。

什么是算法模型？算法建模，包括：机器学习、数据挖掘、人工智能等，也就是现在称为机器学习的方法。大约是上世纪 70 年中后期代开始，以神经网络为前导。特点：根据数

据训练一个算法模型（模型是一个计算机程序，而不是一个数学公式）判断模型的好坏：用一部分不参加建模的数据（测试集）检验另一部分数据（训练集）建立的模型，即：交叉验证。

如果我们把参数模型和算法模型进行比较，那么，参数模型是用数据公式进行描述，先有形式之后，通过训练得到参数估计；算法模型是计算机程序，具有各式各样不同的算法，通过数据进行训练，得到最适合的模型。第四章是参数模型的基础，本章是算法模型的基础。具体的比较要在第七章。

数据建模的重点是问题驱动，而不是（参数）模型驱动或者算法（模型）驱动。用大白话说，我们不要拿着锤子打钉子，而是应该看问题在哪，所以需要什么工具（包括锤子在内），再决定怎么是敲还是打。

所以，整个建模，可以分为三个重点：

首先是找到问题：接着才去采集数据，当然可以是数据挖掘，但也需要定义问题；

其次是拥有数据，接着才会寻找模型，而不是抱着模型找数据：

（一）对于一个数据集需要尽可能尝试各种模型；

（二）分析之前，不应该指定某个模型，而是经过训练后，通过交叉验证（可以比较任何模型），挑选相对较好的模型，进行拟合。

最后，如果在无法证实正态性假定下，往往采用拟合优度检验等进行比较。

二、误解误用

数据建模通常发生误解误用的情况，主要有三种类型：方向性、操作性和推论性。

典型的方向性错误，就是拿着锤子找钉子：研究人员只有考虑一种他熟悉的方法，而没有比较其他模型和交叉验证，就建立了用户模型；研究人员应该避免没有根据或比较地任意假定模型的这种研究过程和形式。

具体而言，两种极端作法造成的方向性错误，包括：

（一）使用或者套用现成模型，或者凭空想象出一套理论；研究人员需要认清的科学研究的意义，在本质上是：任何一个现成模型必须得经受最新数据的考验，否则必须被替代。

（二）隐藏建模过程、数据、方法，甚至隐藏自己的论文，只发布最终结论和建议。

操作性错误，则是主要发生在领域知识以及该领域的科学方法的不够熟悉或者误用。

具体而言，一种是对于“数据”的理解和用法，出现不当；另一种是对于“科学”的理念不够敬畏，总以“走个流程”的态度马马虎虎成事。例如，有人使用宏观数据（任何宏观结论应有微观分析的支持）或者二手数据（均值、百分数、比例）等加工过的数据，不是不可以，而是这些数据仅能提供开始进行研究之前的背景情况了解，而不是作为探索知识或者解决问题所依赖的数据。再如，毫无根据地假定正态性分布，或者没有说明分布，但在数据分析之后，突然出现 t 检验、F 检验和 p 值等；所谓的正态性分布的假定，必须要有理论或是事实上根据。

推论性错误，主要是经过学习数据科学的技巧之后，未与领域学科所积累的知识和经验相结合，未予思考过哲学、科学、工程（见前第一章第一节）而是追求“尽快了事”式的报告产出，或者“尽快发表”式的论文研究而非研究论文。

（一）使用小样本，推论大样本。如果小样本不代表全体，如果大样本不代表全体，那么，小样本也不能代表大样本，那么到底在做什么其实并不知道，但是又很快跳跃式思考，作出了图像结果（见第五章第一节的内容）然后得到若干结论。

（二）数据经过层层“清洗”和加工，与实际实验或者观测对象的距离越来越远（参考

三章第二节）。

（三）毫无根据地提出结论和政策建议。如果我们比较形象一点的说：科学研究的八股文范式成为拍脑袋决策的皇帝新衣，那么这样就不适合了。

表 5.1 常见的数据建模的问题

	说明
方向性	套用模型、工具、公式。 遮掩思路、数据、过程。
操作性	探索和定论的混淆。 假定和假设的混淆。 理论和证据的呼啸。
推论性	过于重视流程：重形式，而不重视研究对象本身。 过于工程思维：重度数据加工，忽视问题本身。 写论文导向的可怕后果：不做研究只看成果的后果。

如果我们把第 2、3、4 章的内容结合，可以得到四个数据建模的基本步骤：

（一）数据采集

（二）数据处理

（三）模型选择

（四）模型判断

然而，上述步骤里，每个环节都有需要仔细小心的地方。简言之，注意陷阱！

三、模型选择

从科学层面而言，模型选择主要依据所要研究的对象和问题，而非模型本身。就数据科学的层面而言，首先仍然是遵循科学，即：研究问题和对象，其次是根据研究对象的数据，进行模型选择。变量属于何种类型的数据（参考第四章第一节），数值变量或者类别变量，确定之后，再看是否具有时间上以及观测次数上的情况，最后区分究竟属于探索性或者验证性，即：构建一个具有理论意义上的模型原型，还是确认以及修正已有的理论或者模型。根据上述三种构面（Aspects），我们可以进行各种已知模型的异同，从而进行大致上的分类，进而找到合适的模型以及知晓运用模型的前提条件。如果遇到新的问题，难以采取已有模型进行研究的情况，再做一番确认之后，发现需要自己建立新的模型，也可据此援引或者找到新的模型的准确定位。

我们考虑常见的横截面数据，如不考虑时间序列以及多次观测的情况下，通常选择的模型，可以分为两大类，即：因变量为数量变量的模型，以及因变量为类别变量的模型。

就因变量为数量变量的情况下，往往最先、最常、最优考虑的是【线性回归】（初步作法见第四章第三节），然而，该模型需要满足以下条件：

（一）必须假定因变量为自变量的线性组合形式；

（二）必须假定可加误差项；

（三）必须假定误差独立同分布；

（四）必须假定损失函数是对称的二次函数。

除了线性回归的模型之外，有时也需考虑【指数变换】和【共线性问题】，后者包括:Lasso、PLS、岭回归等。

如果线性回归、指数和共线性的考虑之后，有些数据是用【多项式回归】模型和【非线性回归】的方式，才能分析得出结果，然而，我们就有必要深度理解其函数形式了。

以上，是经典统计分析的主要方法，这些模型的选择，除了数据本身的特征（如横截面

数据并且因变量是数值变量）以外，主要是依据数学推导所建立的理论模型，由数据和模型进行验证。如果涉及第七章的机器学习方法，在横截面数据上，则有以下几种模型，可供参考：

（一）决策树（回归树）：在 R 里，有 rpart 和 tree 等程序包。

（二）Boosting 回归：在 R 有 mboost 等程序包。

（三）Bagging 回归：在 R 有 ipred 等程序包。

（四）随机森林回归：在 R 有 randomForest 等程序包。

（五）人工神经网络回归：在 R 有 nnet 等程序包。

（六）支持向量机回归：在 R 有 rminer 等程序包。

以上，机器学习模型，可以全部采用之后，通过 K 折交叉验证进行模型比较，选择适合的模型，其思路与之前介绍的经典统计模型有所不同。

以下，我们仍然讨论横截面数据，但，如果因变量不是数值变量，而是因变量为类别变量的时候，那么，哪些模型合适？

通常，我们考虑两种情况：第一种是 logistic 和 probit 回归（因变量不仅是类别变量，而且还是二分变量的情况），此时我们可以借用 R 的 glm 程序包。第二种是判别分析（因变量是类别变量，而且自变量必须是数值变量），此时我们可以借用 R 的 lda 程序包。同理，这些是经典统计模型，需要从学科领域的理论进行数学推导之后，建立数学模型再以数据和算法进行验证。

那么，我们同样可以采取机器学习（参考第七章）的诸多模型，在没有上述因变量的变量数值类型或者自变量的条件下，采取各种模型进行计算之后的比较。这些模型有：

（一）决策树（分类树）：在 R 有 rpart 和 tree 等程序包。

（二）Adaboost 分类：在 R 有 adabag 等程序包。

（三）Bagging 分类：在 R 有 adabag 等程序包。

（四）随机森林分类：在 R 有 randomForest 等程序包（在第四章第三节中曾经介绍）。

（五）支持向量机分类：在 R 有 e1071 等程序包（不同版本的 R 可能不同）。

（六）最近邻方法分类：在 R 有 kknn 等程序包（在第七章会重点介绍）。

前面提过，因变量是数值变量，可以采取经典统计模型以及机器学习模型两种思路，但究其主要分析目的，或者说是模型用途，是为了解决【回归问题】，解释以及预测，如果增加新的样本数据，能够带来什么短期趋势或者结果。如果因变量是类别变量，则是为了解决【分类问题】，即能够判断新的样本在我们所建立的模型里，会被分类为哪一种情况的问题。

在回归问题的求解过程，有遇到指数或者多项分布的情况，此处，在分类问题上，如果遇到指数或者多项分布的情况，可以采取以下几种模型。

我们可以考虑 Poisson 对数线性模型，在 R 有 glm 等程序包，近年来一直还有其他程序包，提供解决各种新的“例外”情况的便捷工具。例如，如果遇到 Dispersion 现象，可以采用 dglm、statmod 和 glm.nb 等进行分析，如果遇到零膨胀现象，可以采取 pscl 等程序包，进行分析。

在多项分布情况下，可以考虑【多项 logit 模型】和【多项分布对数线性模型】两种机器学习方法，把分类问题利用回归模型进行处理，前者在 R 有 mlogit 等程序包，后者在 R 有 MASS 和 nnet 程序包，可供利用使用。

以上，主要是横截面数据的两种情况，即：因变量是数值变量还是类别变量、所要处理的是回归问题还是分类问题、所要应用的手段是经典统计还是机器学习等的各种考虑。

那么，除了常见的横截面数据之外，我们有可能面对的是纵向数据、多水平数据、面板数据等；在这些不一定常见的模型当中，相对常见的模型，有以下几种：

（一）线性随机效应混合模型：在 R 有 nlme 和 lme4 等程序包。

（二）广义线性随机效应混合模型：在 R 有 lme4 程序包。
（三）决策树及随机效应模型，在 R 有专门的 REEMtree 等程序包。
（四）纵向生存数据：在 R 有 coxme 和 JM 等程序包。而且，
（五）一般生存分析：在 R 有 survival 程序包。以及，
（六）面板数据：在 R 有 plm 等程序包。

以上，属于横截面数据之外的几种情况，但还不能算是所有情况。如果我们对于因变量和自变量不作区分，那么，我们还有【多元分析】的情况值得考虑，首先，满足多元正态分布数据的条件，接着模型选择，经典统计的多元分析，包括：

（一）主成份分析；
（二）因子分析；
（三）聚类分析；
（四）相关分析等。

上述模型，有时可以采取混合研究方法的方式，依次进行或者调整次序后进行分析。此外，在非数值变量的情况下的【对应分析】可以分层和分群，例如:
（一）现代多元分析：在 R 有 FactoMineR 等程序包。
（二）关联规则分析：在 R 有 arules 等程序包。

如果既不是横截面数据，也不是纵向数据、多水平数据、面板数据，在方法上即不属于经典统计理论也不属于机器学习理论，又不能以多元分析和对应分析从数据中得到有价值的信息，那么，还有两种模型可供考虑：
（一）路径分析模型：例如 PLS 方法。在 R 有 plspm 等程序包。该模型不要求多元正态分布，而且可以求出潜变量的值，并且样本的数据量可以少些。然而，这个模型有赖领域知识的支撑，而非完全从数学推导与假设检验、机器学习与交叉验证而来，它是一种通过数据拟合进行理论建构的活动。
（二）协方差方法：例如结构方程模型，在 R 有 sem 等程序包，在 SPSS 软件有 AMOS 模块，但其实最早是 LISREL 语言（以及软件）和 ML 软件。该模型要求多元正态分布数据，区分为探索性和验证性，但均不够稳健，该模型不能求出隐变量的值和绝对系数。不过，因为有些研究问题无法或者不适合不便于直接测量，需要通过测量其他方面之后，在学科理论指导下假定不存在的变量存在（潜变量），所以协方差方法，包括结构方程模型或者层次潜变量模型，在一些情况下，反而成为常见的模型选择选项。

四、实施要点

在本节第二部分我们介绍了可能误解误用的几种情况，便于第三部分在考虑具体研究问题、数据和模型选择时，能够根据数据本身、分析目的与方法，以及条件与限制等进行模型选型。根据数据建模的四个步骤：

（一）数据采集；
（二）数据处理；
（三）模型选择；
（四）模型判断。

我们需要针对每个环节可能发生的陷阱，或者说是注意事项，进行说明，接着才有问题、数据、算法、模型的介绍，否则容易被认为是单纯的算法和代码的讲解，这样就失去了数据科学的本意和乐趣了。因此，本部分将对第一部分的内容进行更为细致的说明。

数据采集 数据处理 模型选择 模型判断

图 6.2 数据采集

在数据采集的环节上，首先，根据实际目的进行收集：有些是人工采集，例如：问卷调查、访谈调查、观察记录、实验室采集等手段获得；有些是通过信息系统，比如：日志数据、查询数据、系统日志、用户日志、查询输入数据等。

其次，确定哪些变量的数据需要收集，这与数学、应用统计或者信息系统架构无关，而是对行业领域的了解程度和经验。 再者，不是有了数据，就要马上得到所需要的结论，而是取得：与所关心的问题直接有关的变量的数据。

最后，有一个重要观念是：质量差的数据得不到高质量的结论。



图 5.3 数据处理

在数据处理的环节上，原始数据往往或多或少地存在各种缺失值，以及不合逻辑或不一致等的情况，这就需要预处理方案。这类工作很可能耗时而且琐碎，但必须完成，否则无法后续分析或者造成的负面影响难以估量。

此外，在第三章第三节和第四章第三节，我们介绍了一些处理缺失值的方法：

（一）整条数据及其所有变量的数据均删除；

（二）用同一变量其他值的均值或中位数填补；

（三）在各个变量之间建立模型（比如回归模型、最近邻方法、随机森里等）进行填补。

如何处理缺失值，主要依据行业的领域知识与科研经验，以及所研究的研究对象，以及研究设计来决定。如果缺失值的填补，不影响造成主要结论的变量或者列，或者虽然造成影响但是能够估算出影响范围的大小，如误差上下标准，则可以选择适合的缺失值处理方法，否则需要谨慎小心，避免出现为了解决缺失值问题造成的套套逻辑的研究。

数据采集 数据处理 模型选择 模型判断

图 5.4 模型选择

在模型选择的环节上，比较复杂，也是本节第一部分到第三部分反复强调的重点。

首先需要明确的是，建模目的在于：预测、解释，或者理解产生数据，因此目的不同，所采取的选择标准和判断方式的机制就有不同。

其次，如果是探索性分析，可以通过几种方式：

（一）利用图形；

（二）描述统计；

（三）探索分析方法，如：关联性、线性、异方差性、多重共线性、聚类特征、平衡特征、分布形状等。

然而，依靠工具和方法是一部分，另一部分是科研人员的直觉。这是很重要的能力，也是判断科研人员的科研能力的一种展现，科学研究的“工业流水线”很容易学习和掌握，但直觉则是需要长期锻炼和积累（想得越多而且做的越多，能力越大，和年龄无关）。当然，这种直觉，或者说是猜想，也是有所章法的，例如，寻找现成的模型，可有几个步骤：

（一）比较各种模型的计算结果；

（二）如果现有模型不能满足需要，就可能产生新的分析方法；

（三）模型选择的过程贯穿于整个数据分析过程。

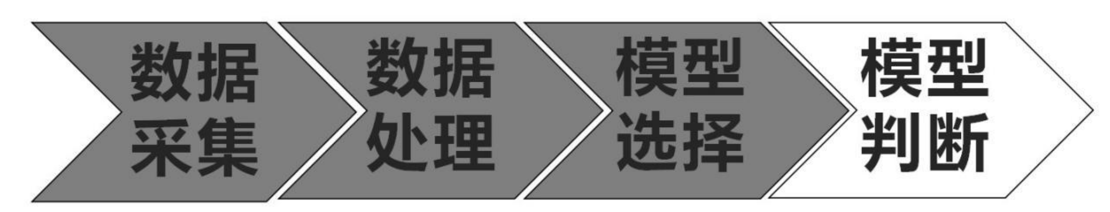


图 5.5 模型判断

在模型判断的环节上，需要注意：假定，交叉验证，以及误差，三个重点。这是因为选择模型不是最终目的，最终目的是解释模型所产生的结果；结果必须是应用领域的结果，需要具有实际意义；仅仅用统计术语或者计算机术语说某个模型较好，缺乏行业知识和判断，不是可靠的模型选择。

假定，是源于信仰或者方便；在科学研究中，也有貌似不科学的一面，假定的选择和研究人员的直觉有关。（我们这里指的不是随意凭感觉，而是凭经验，拍脑袋也有其道德高标准，包括何时应该使用何时应该慎用这种能力的选择，也是直觉的一部分）。

传统上，进行应用统计时，通常需要假定分布（如正态性）和模型，并且决定损失函数，由此制定检验值和临界值的判别准则。假定，它无法用确定性方法被验证，仅能尝试采用显著性检验来拒绝假定的正反两面，因此把假定换个词叫假设；然而，即使没有充分理由否定反面，也不能“证明”正面是正确，它仍然属于假定。

交叉验证（cross validation）适用于传统模型之间或者在传统模型和算法模型之间的比较，它的主要作法包括了：

（一）拿一部分数据作为训练集（training set）得到模型；

（二）再用另一部分数据（称为测试集 testing set）来看误差。

（三）有时，需要进行 k 折交叉验证（k-fold cross validation）：把数据分成 k 份，每次拿 k-1 份作为训练集，用剩下的一份作为测试集，重复 k 次，得到 k 个误差作出平均，以避免仅用一个测试集可能出现的偏差。

值得注意的是：算法模型，由于没有传统模型的那些假定，所以判断模型优劣通常仅用交叉验证。

最后谈谈【误差】：误差越小，模型越优，建模意义越大，学说的可靠性越强。

误差，有两部分组成：

（一）研究果在假定的模型之下的精确度，包括：相合性、无偏性等。

（二）假定的模型和真实的规律之间的距离。

如果是问题驱动和数据驱动的研究人员，就能够回答上述第二个问题，但是如果是以模型驱动的研究人员只能选择回避第二个问题。如果您能够明白这件事情，那么就不难明白我们所说的数据科学，与过去常规的科学研究，有什么不太一样的地方和拓展之处了。

第二节 排名

一、社会网络

接下来，我们不打算把第一节第三部分的各种模型进行拓展，反而是回到第四章第三节的社会网络图的讨论，而且我们接下来采取 Python 编程的方式进行。因为我们在第六章以及之后，要能把所有知识点结合起来，既能选择合适的理论与模型，又能读懂和修改需要使用的 R 或者 Python 的代码，为我们自己开展数据科学的相关研究。

以 Python 编写的网络分析的示例代码，虽然是一整段代码，但是为了便于理解以及修改成为日后我们可以使用操作的代码，我们分为五个部分：

（一）处理向量的函数：为（三）建立好分析所需的公式。
（二）处理矩阵的函数：为（三）建立好计算所需的工具。
（三）数据准备：如果本部分对您只是”另一套“傻瓜软件，那您更替数据，即可分析。
（四）准备算法：实际上的社会网络算法。
（五）计算用户的社会网络的值：得出并且打印呈现计算结果的数值。

```
import math, random, re
from collections import defaultdict, Counter, deque
from functools import partial
# 导入了数学计算的 math 库以及文本处理的 re 库

# 1.处理向量的函数
# 1.1【两对象相乘相加】：公式为 v_1 * w_1 + ... + v_n * w_n
def dot(v, w):
    return sum(v_i * w_i for v_i, w_i in zip(v, w))

# 1.2.【逐个减去两个向量】
def vector_subtract(v, w):
    return [v_i - w_i for v_i, w_i in zip(v,w)]

# 1.3.【逐个促使两个向量相加】
def vector_sum(vectors):
    return reduce(vector_add, vectors)

# 1.4.【逐个促使两个向量相乘】
def scalar_multiply(c, v):
    return [c * v_i for v_i in v]

# 1.5.【向量均值】：计算其第 i 个元素，输入向量中第 i 个元素的均值的向量。
def vector_mean(vectors):
    n = len(vectors)
    return scalar_multiply(1/n, vector_sum(vectors))

# 1.6.【平方和】：公式为 v_1 * v_1 + ... + v_n * v_n
def sum_of_squares(v):
    return dot(v, v)
```

```
# 1.7.【规模】：平方和开根号。
def magnitude(v):
    return math.sqrt(sum_of_squares(v))

# 1.8.【距离平方】
def squared_distance(v, w):
    return sum_of_squares(vector_subtract(v, w))

# 1.9.【距离 distance】：距离平方的开根号。
def distance(v, w):
   return math.sqrt(squared_distance(v, w))
```

上述代码，此处，我们利用 Python 所提供的【def 函数】把所需要做的事情（功能），变成 Python 能够调用的一个我们自己建立的【函数】或者一组【模块】；多个函数或者模块可以叠加起来一起使用。使用 def 时，首先需要定义函数，首先是【函数名】并且在函数名的括号内，指定【参数】。接着，在函数内部，也就是冒号之后，指定具体怎么实现（如何进行）计算的代码。最后，通常*（不是一定）末尾加上【return 函数】将函数经过计算的【值】返回。

```
# 2.处理矩阵的函数
# 2.1.【形状】：返回 A 有多少行、多少列。
def shape(A):
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0
    return num_rows, num_cols

# 2.2.【取行】：从 A 矩阵获得 i 行。
def get_row(A, i):
    return A[i]

# 2.3.【取列】：从 A 矩阵获得 j 列。
def get_column(A, j):
    return [A_i[j] for A_i in A]

#2.4.【创建矩阵】：返回行数乘以列数的矩阵，以第（i,j）个实体（entry）作为该矩阵的实体（i,j）。
def make_matrix(num_rows, num_cols, entry_fn):
    return [[entry_fn(i, j) for j in range(num_cols)]
            for i in range(num_rows)]

# 2.5.【对角线】：位于对角线上是 1，其余为 0
def is_diagonal(i, j):
    return 1 if i == j else 0
```

上述代码，需要理解【循环逻辑】以及正则表达式的 for 循环语法。

```
# 3.数据准备
# 3.1 用户信息
```

```
## 定义矩阵是5乘5的矩阵，以对角线为矩阵实体。（见2.4和2.5）
identity_matrix = make_matrix(5, 5, is_diagonal)

## 用户编号 id 以及名称 name
users = [
    { "id": 0, "name": "Alan" },
    { "id": 1, "name": "Mark" },
    { "id": 2, "name": "Lisa" },
    { "id": 3, "name": "Irina" },
    { "id": 4, "name": "Eirc" },
    { "id": 5, "name": "Karl" },
    { "id": 6, "name": "Rose" },
    { "id": 7, "name": "Berna" },
    { "id": 8, "name": "Joe" },
    { "id": 9, "name": "Kim" }
]
```

如果 6.2.1 对您只是”另一套“傻瓜软件，那您更替您所需要的数据，在此，我们建立的是一组组【键值对】的数据，这类数据便于分析，我们可以采取导入数据的方式（见 2.1.）。

```
# 3.2 关系信息
## 根据3.1的用户编号，建立两两关系。
friendships = [(0, 1), (0, 2), (0, 3), (0, 4), (3, 6), (3, 7),
               (2, 6), (2, 7), (5, 7), (6, 8), (4, 8), (8, 9)]

# 赋予每位用户一张关系列表。
for user in users:
    user["friends"] = []

# 填充列表
for i, j in friendships:
    # 此处users[i]是该为用户，当它的id号为i时。
    users[i]["friends"].append(users[j]) # 添加 i 作为j的朋友。
    users[j]["friends"].append(users[i]) # 添加 j 作为i的朋友。
```

此处，是把上述【键值对】的数据中，涉及到【键】的部分，进行两两关联，形成一个网络关系，其作法类似我们在 5.3.3 里所做的事情。

```
# 4.准备算法
# 4.1.中间性（Betweenness Centrality）
def shortest_paths_from(from_user):
    # 该用户（user_id）与所有用户的最短路径的字典。
    shortest_paths_to = { from_user["id"] : [[]] }
    # 用以检查（上一个用户，下一个用户）的队列（queue）。
    # 启动所有配对（用户、用户的朋友）。
    frontier = deque((from_user, friend)
                     for friend in from_user["friends"])
    # 使用while循环，持续配对直到清空队列为止。
    while frontier:
        # 从头开始，从用户 id 的第一位到最后一位。
```

```
        prev_user, user = frontier.popleft()
        user_id = user["id"]
        # 往 prev_user 的最短路径，作为 user 连接 prev_user 的路径。
        paths_to_prev = shortest_paths_to[prev_user["id"]]
        paths_via_prev = [path + [user_id] for path in paths_to_prev]
        # 已知到达此处的最短路径
        old_paths_to_here = shortest_paths_to.get(user_id, [])
        # 求：目前为止的最短路径。
        if old_paths_to_here:
            min_path_length = len(old_paths_to_here[0])
        else:
            min_path_length = float('inf')

        # 任何新的路径到达此处：尽量使得这些路径不长。
        new_paths_to_here = [path_via_prev
                             for path_via_prev in paths_via_prev
                             if len(path_via_prev) <= min_path_length
                             and path_via_prev not in old_paths_to_here]

        shortest_paths_to[user_id] = old_paths_to_here + new_paths_to_here
        # 为用户增加新的朋友：在最短路径上没有看到的朋友，添加到网络图的边界上。
        frontier.extend((user, friend)
                        for friend in user["friends"]
                        if friend["id"] not in shortest_paths_to)
    # 返回最短路径。
    return shortest_paths_to

for user in users:
    user["shortest_paths"] = shortest_paths_from(user)

for user in users:
    user["betweenness_centrality"] = 0.0

for source in users:
    source_id = source["id"]
    for target_id, paths in source["shortest_paths"].items():
        if source_id < target_id:   # 不做重复计算
            num_paths = len(paths)  # 计算有多少最短路径
            contrib = 1 / num_paths # 提供数值给特征向量中心性
            for path in paths:
                for id in path:
                    if id not in [source_id, target_id]:
                        users[id]["betweenness_centrality"] += contrib
```

此处，语法上没有更多新的内容，主要不够熟悉的可能是【中间性】的含义，以及代码如何反映了中间性的数学公式。

```
# 4.2.亲密性（closeness centrality）
# 【距离 farness】：到每位其它用户的最短路径的长度的和。
```

```
def farness(user):
    return sum(len(paths[0])
               for paths in user["shortest_paths"].values())

for user in users:
    user["closeness_centrality"] = 1 / farness(user)
```

中间性，计算：该用户与所有用户的最短路径。亲密性，计算：该用户到每位用户的最短路径的长度和。这两个计算都有赖于一开始的向量计算的 def 函数。

```
# 4.3.特征向量中心性（eigenvector centrality）
# 4.3.1. 特征向量中心性的核心：矩阵乘法（matrix multiplication）
def matrix_product_entry(A, B, i, j):
    return dot(get_row(A, i), get_column(B, j))

# 4.3.2.【矩阵乘法】
def matrix_multiply(A, B):
    n1, k1 = shape(A)
    n2, k2 = shape(B)
    if k1 != n2:
        raise ArithmeticError("incompatible shapes!")
    return make_matrix(n1, k2, partial(matrix_product_entry, A, B))

# 4.3.3.【作为矩阵的向量】：返回把向量 v（以列表形式）作为 n x 1 的矩阵。
def vector_as_matrix(v):
    return [[v_i] for v_i in v]

# 4.3.4.【作为矩阵的值列表】返回数值形成的列表形式的 n x 1 的矩阵。
def vector_from_matrix(v_as_matrix):
    return [row[0] for row in v_as_matrix]

# 4.3.5.【矩阵运算子】
def matrix_operate(A, v):
    v_as_matrix = vector_as_matrix(v)
    product = matrix_multiply(A, v_as_matrix)
    return vector_from_matrix(product)

# 4.3.6.【特征向量】
def find_eigenvector(A, tolerance=0.00001):
    guess = [1 for __ in A]
    while True:
        result = matrix_operate(A, guess)
        length = magnitude(result)
        next_guess = scalar_multiply(1/length, result)
        # 如果估计和估计值小于容器（tolerance）则返回估计值（特征向量）和长度（特征值）
        if distance(guess, next_guess) < tolerance:
            return next_guess, length
        # 估计 = 特征向量
        guess = next_guess
```

```
# 4.3.7【征向量中心性的公式】
def entry_fn(i, j):
    return 1 if (i, j) in friendships or (j, i) in friendships else 0
n = len(users)
adjacency_matrix = make_matrix(n, n, entry_fn)
eigenvector_centralities, _ = find_eigenvector(adjacency_matrix)
```

特征相量中心性，采取矩阵计算的方式，便于降低计算资源（提高计算效率），它有赖之前 def 的矩阵函数。

```
# 4.4. PageRank 算法
# 4.4.1.PageRank 算法的核心：有向图（directed graphs）
# 链接（链入和链出）的对值
endorsements = [(0, 1), (1, 0), (0, 2), (2, 0), (0, 3), (3, 0), (1, 3),
                (1, 2), (3, 4), (5, 4), (5, 6), (7, 5), (6, 8), (8, 7), (8, 9)]

for user in users:
    user["endorses"] = []       # 增加一项追踪 endorse（链出）的列表。
    user["endorsed_by"] = []    # 另一项追踪 endorse（链入）的列表。

for source_id, target_id in endorsements:
    users[source_id]["endorses"].append(users[target_id])
    users[target_id]["endorsed_by"].append(users[source_id])

endorsements_by_id = [(user["id"], len(user["endorsed_by"]))
                      for user in users]

sorted(endorsements_by_id,
       key=lambda pair: pair[1],
       reverse=True)
```

关于 PageRank 算法，我们在本节第三部分和第四部分进行逐一讲解。

```
# 4.4.2.PageRank 算法的公式
# 目前设定的 PageRank 计算方式：导入用户数据，阻尼系数 0.85，迭代 100 次。
def page_rank(users, damping = 0.85, num_iters = 100):
    # 均匀地散播最初的 PageRank 值
    num_users = len(users)
    pr = { user["id"] : 1 / num_users for user in users }
    # 各个节点（node)获得各自迭代，属于 PageRank 的一部分。
    base_pr = (1 - damping) / num_users
    for __ in range(num_iters):
        next_pr = { user["id"] : base_pr for user in users }
        for user in users:
            # 传输 PageRank 值到链出的值
            links_pr = pr[user["id"]] * damping
            for endorsee in user["endorses"]:
                next_pr[endorsee["id"]] += links_pr / len(user["endorses"])
```

```
        pr = next_pr
    return pr
```

最后，打印计算结果。

```
# 5. 计算用户的社会网络的值
if __name__ == "__main__":
    print("用户的社会网络：10 位用户按照四种计算方式，取得的两两相对的值")
    print("中间性指标 Betweenness Centrality")
    for user in users:
        print(user["id"], user["betweenness_centrality"])
    print()

    print("亲密性指标 Closeness Centrality")
    for user in users:
        print(user["id"], user["closeness_centrality"])
    print()

    print("特征向量中心性 Eigenvector Centrality")
    for user_id, centrality in enumerate(eigenvector_centralities):
        print(user_id, centrality)
    print()

    print("排名指标 PageRank")
    for user_id, pr in page_rank(users).items():
        print(user_id, pr)
```

结果显示 10 位用户，根据四种算法，得到的排名数值，如下：

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
7 8.333333333333334
8 9.0
9 0.0

亲密性指标Closeness Centrality
0 0.0625
1 0.041666666666666664
2 0.0625
3 0.0625
4 0.05263157894736842
5 0.03571428571428571
6 0.058823529411764705
7 0.05
8 0.05263157894736842
9 0.037037037037037035

特征向量中心性Eigenvector Centrality
0 0.4443302804338586
1 0.1581588935020989
2 0.419308046405739
3 0.419308046405739
4 0.2514990914743666
5 0.12167024726770725
6 0.3918507305473259
7 0.34181985198996295
8 0.26222557054858964
9 0.09334019797226772

排名指标PageRank
0 0.09601406799537791
1 0.042203985932029786
2 0.05416178194610715
3 0.05416178194610715
4 0.0564825538045953
5 0.04344422700587085
6 0.03346379647749512
7 0.03346379647749512
8 0.04344422700587085
9 0.03346379647749512
>>> |
Ln: 53 Col: 4
```

图 5.6 社会网络分析结果（Python）

事实上，这是一种排名次的游戏（根据数值大小得来）；换言之，这种排名结果的顺序，是考虑到了该用户在群体之中的社会网络关系，最终作为优先次序的依据或者参考。

二、排名算法

因为在前面的社会网络分析当中，我们用到了比较著名的 PageRank 算法，并且相较于中间性、亲密性和特征向量中心性，还是比较复杂的一套计算方式和思想。因此本节对其基本思想，进行简要介绍。

首先，它的基本思路，包括以下三个重点：

（一）网页的重要性由网页间的链接关系所决定；

（二）根据网页之间的链接结构，评价每个页面的重要性（等级）；

（三）一个网页的 PR 值不仅考虑指向它的链接网页数，还有指向“指向它”的网页的其他网页本身的重要性。

根据上述这个思路，可以推断需要满足两个假定，作为解释和分析最后计算结果的重要基础，其两大假定的概念定义，如下：

（一）数量：如果一个页面节点接收到的其他网页指向的入链数量越多，那么这个页面越重要；

（二）质量：质量高的页面会通过链接向其他页面传递更多的权重。所以越是质量高的页面指向页面 A，则页面 A 越重要。

上述，是科学和哲学层面的讨论，我们要用做数据科学的”实践“的话，必须考虑工程问题，即如何实现。那么，如果我们要计算 PageRank 就要把握三个重点：

（一）反向链接数；

（二）反向链接是否来自 PageRank 较高的页面；

（三）反向链接源页面的链接数。

具体公式，如图 5.7 所示。

$$PR(p_i) = \frac{1-d}{n} + d \sum_{p_j \in M(i)} \frac{PR(p_j)}{L(j)}$$

图 5.7 PageRank

PR(pi): pi 页面的 PageRank 值

n: 所有页面的数量

pi: 不同的网页 p1,p2,p3

M(i): pi 链入网页的集合

L(j): pj 链出网页的数量

d:阻尼系数（设为 0.85 ），用户到达某页面后并继续向后浏览的概率。

以上，为 PageRank 的简要介绍，以下，我们利用 R 进行编程示例。

```
# 建立一组试验数据，此时先做一个空的数据框。如果我们采取 2.1.1 的作法会太复杂。
packagerank <- data.frame()
# 此处选择【R Console】的【编辑】的【数据编辑器】
write.csv(package,"C:/Users/Hp/Desktop/package.csv",row.names=FALSE)
# 写入.csv 文件。
```

```
pages<-read.table(file="C:/Users/Hp/Desktop/packagerank.csv",header=T,sep=",")
# 读取.csv 文件,注意此时 header = T 因为我们接下来要写入【表头】（变量名称）
names(pages)<-c("src","dist")
# 命名变量名称
page
# 查看数据
```

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R是自由软件，不带任何担保。
在某些条件下你可以将其自由散布。
用'license()'或'licence()'来看散布的详

R是个合作计划，有许多人为之做出了贡献.
用'contributors()'来看合作者的详细情况
用'citation()'会告诉你如何在出版物中正

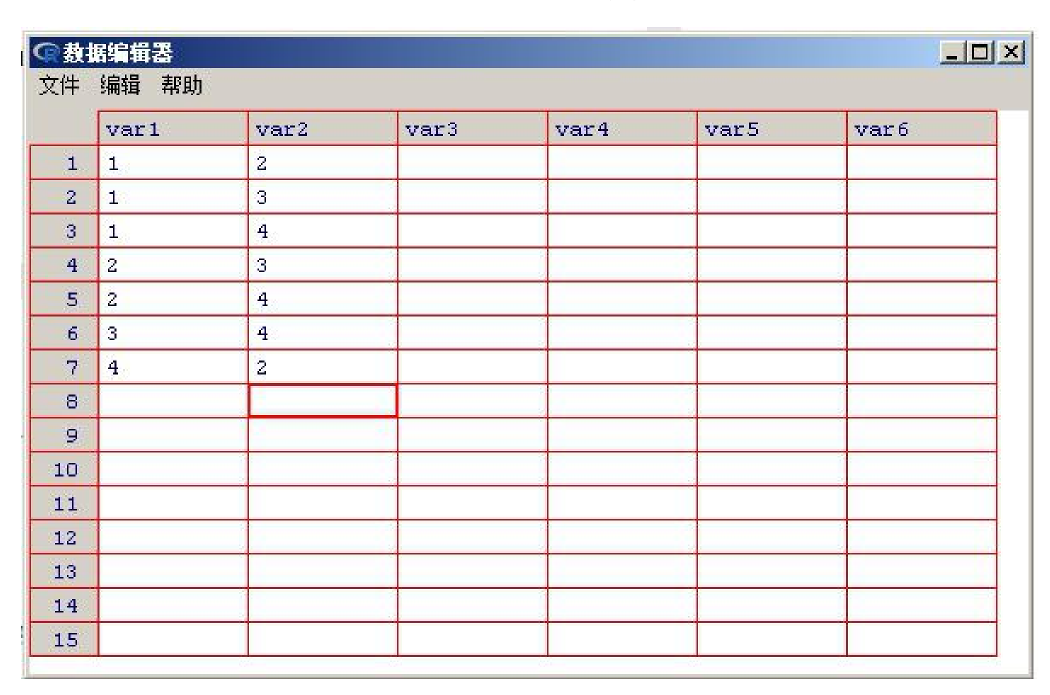
用'demo()'来看一些示范程序，用'help()'
用'help.start()'通过HTML浏览器来看帮助
用'q()'退出R.

[原来保存的工作空间已还原]

> packagerank <- data.frame()
>

Question
数据框或矩阵名
packagerank
确定 取消
```

图 5.8 使用数据编辑器（R）



数据编辑器
文件 编辑 帮助

	var1	var2	var3	var4	var5	var6
1	1	2				
2	1	3				
3	1	4				
4	2	3				
5	2	4				
6	3	4				
7	4	2				
8						
9						
10						
11						
12						
13						
14						
15						

图 5.9 用数据编辑器编写数据框（R）



图 5.10 数据写入完成（R）

这笔数据的情况，如果我们用 Excel 表来看，如图 6.11 所示

pagerank.csv - Micros...

D10

	A	B
1	1	2
2	1	3
3	1	4
4	2	3
5	2	4
6	3	4
7	4	2
8		
9		

pagerank 100%

页面1链向页面2,3,4
页面2链向页面3,4
页面3链向页面4
页面4链向页面2

图 5.11 PageRank 示例数据（Excel）

示例数据完成之后，我们接下来进行 PageRank 的理解，从做（R）中学（PageRank）的效率较高，我们要做的就是三件事：

（1）邻接矩阵；

（2）概率转移矩阵；

（3）计算 PageRank 值。

首先，邻接矩阵的代码如下。

```
Matrix01<-function(pages){
n<-max(apply(pages,2,max))
A<-matrix(0,n,n)
for(i in 1:nrow(pages))A[pages[i,]$dist,pages[i,]$src]<-1
A
}
# 此处我们使用 apply 函数，取矩阵的最大值。
# 正则表达式的 for, [ ], $, i in 等，我们在上一章的课堂上讲解过。
A<-Matrix01(pages)
A
```

其实，我们已经没有必要一张张截图，求得心理安慰，但是为了有所了解和确认代码结果，所以还是截图如下：

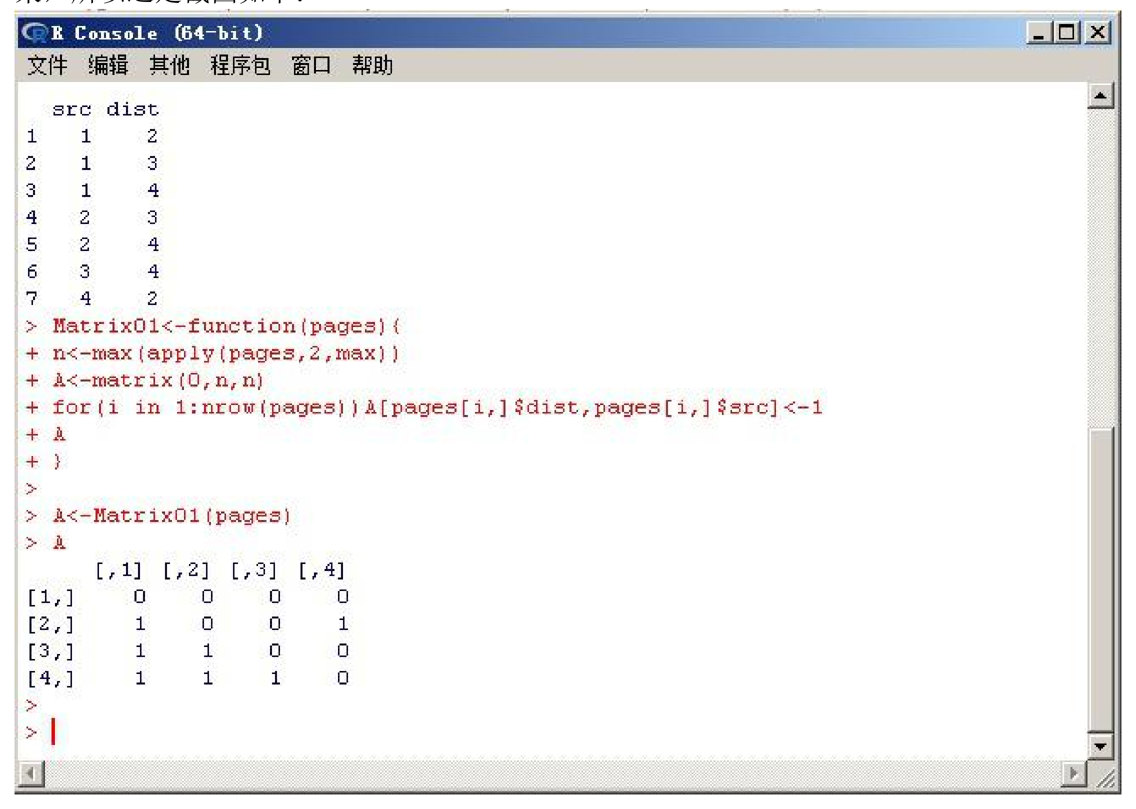


图 5.12 邻接矩阵示例（R）

接着，是概率转移矩阵，在 R 的示例代码，如下。

```
Matrix02<-function(G){
cs<-colSums(G)
cs[cs==0]<-1
n<-nrow(G)
A<-matrix(0,nrow(G),ncol(G))
for (i in 1:n) A[i,]<-A[i,] + G[i,]/cs
A
}
# 此处，仍然是正则表达式 == 代表”等于“的意思，因为 = 已经代表定义（和赋值）了。
# 此处，还使用了 colSums()函数，以及 A[i,] + G[i,]/cs 作为概率矩阵的计算式。
G<-Matrix02(A)
G
```

数据科学采用 Python 的好处是可以快速实现，并且相对容易转化为其他编程语言（如 JAVA 等），而您如果使用 R 则相对容易理解每个步骤和结果：

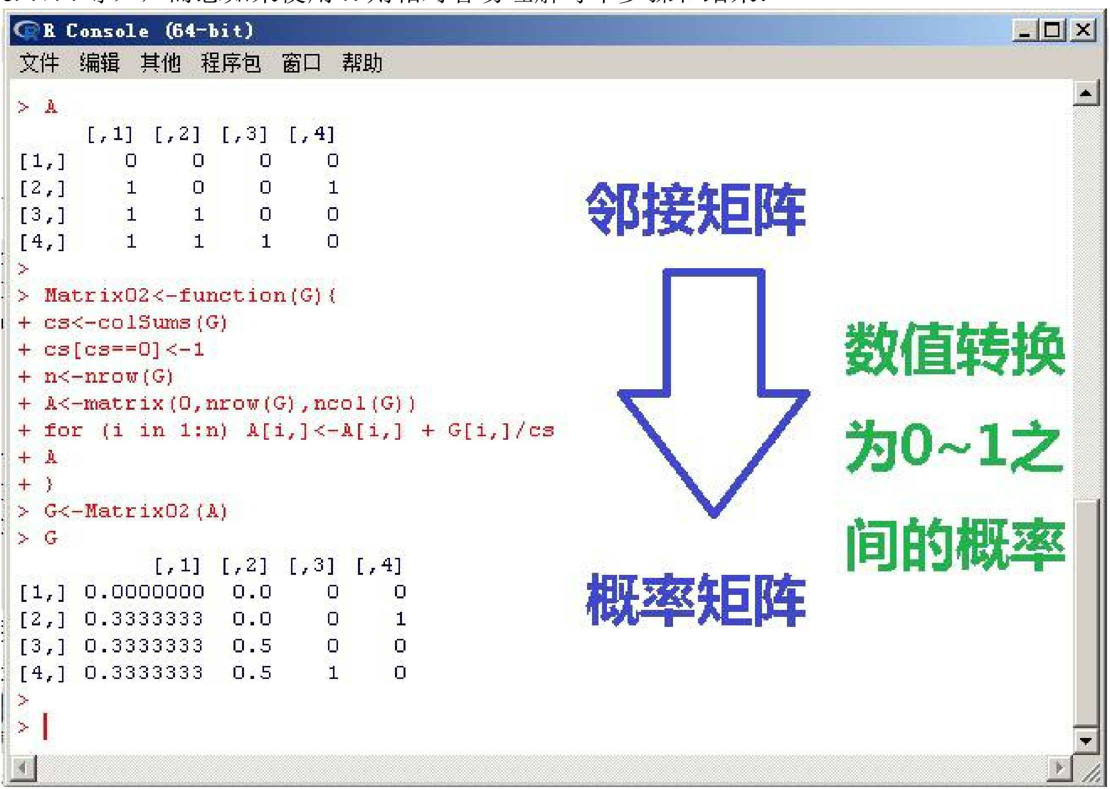


图 5.13 概率矩阵示例（R）

最后，第三步是计算 PageRank 值，示例代码，如下。

```
Matrix03<-function(G,iter=100){
iter<-10
n<-nrow(G)
x<-rep(1,n)
for (i in 1:iter) x<- G %*% x
x/sum(x)
}
# 这里使用函数 rep( )是为了迭代

q<-Matrix03(G,100)
q
```

```
R Console (64-bit)
文件  编辑  其他  程序包  窗口  帮助
> G<-Matrix02(A)
> G
          [,1] [,2] [,3] [,4]
[1,] 0.0000000  0.0    0    0
[2,] 0.3333333  0.0    0    1
[3,] 0.3333333  0.5    0    0
[4,] 0.3333333  0.5    1    0
>
> Matrix03<-function(G,iter=100){
+ iter<-10
+ n<-nrow(G)
+ x<-rep(1,n)
+ for (i in 1:iter) x<- G %*% x
+ x/sum(x)
+ }
>
> q<-Matrix03(G,100)
> q
          [,1]
[1,] 0.0000000
[2,] 0.4036458
[3,] 0.1979167
[4,] 0.3984375
>
```

图 5.14 通过概率矩阵计算结果

图 6.14 的下方的结果，有如下含义：

（1）页面 1 的 PR 值是 0 这表示：没有指向页面 1 的页面。

（2）页面 2 的 PR 值是 0.4 这表示：页面 1 和 4 都指向 2；而且 4 不仅权重高，还只有指向到 2 而已。

（3）页面 3 的 PR 值是 0.19 这表示：虽然 1 和 2 的指向了 3 但是 1 和 2 还指向的其他页面，权重被分散。

（4）页面 4 的 PR 值是 0.39 这表示：权重很高，因为被 1,2,3 都指向。

我们是通过概率矩阵计算结果，而非完整地按照 PageRank 的公式（图 6.7）进行计算得到的结果。然而，我们仍然可以对照图 6.11 的左侧数据示例以及右侧说明，与图 6.13 下方的结果，得到互相印着，说明概率矩阵的计算能够说明我们对于网络链接的认识。

三、算法实现

尽管前面第二部分已经能够通过概率矩阵计算【基于网络关系（有向链接）】的结果，然而，我们尚未考虑到图 5.7 里的 Delta（阻尼系数）。所以，在本部分就主要实现，并且予以比较。

在 R 的示例代码，如下。

```
# 假如我们把那个 pagerank.csv 文件放到计算机的 D 槽，并且过了几天，开始做这件事的话，那么，我们可以通过设置 R 的工作路径，进行导入。
getwd()
setwd("D:/")
# 查看工作路径、设置工作路径。
```

```
pages<-read.table(file="D:/pagerank.csv",header=FALSE,sep=",")
# 因为路径设置在计算机 D 槽，所以我们的指令不像 6.2.2 了。
# 如果您的.csv 文件的内容，如图 5.11 所示，那么 header =F
# 如果您的.csv 文件，来自图 5.9 之后就没有变更过，那么 header= T
names(pages)<-c("src","dist")
page
```

邻接矩阵，如下，与图 5.12 一样。

```
Matrix01<-function(pages){
n<-max(apply(pages,2,max))
A<-matrix(0,n,n)
for(i in 1:nrow(pages))A[pages[i,]$dist,pages[i,]$src]<-1
A
}
A<-Matrix01(pages)
A
```

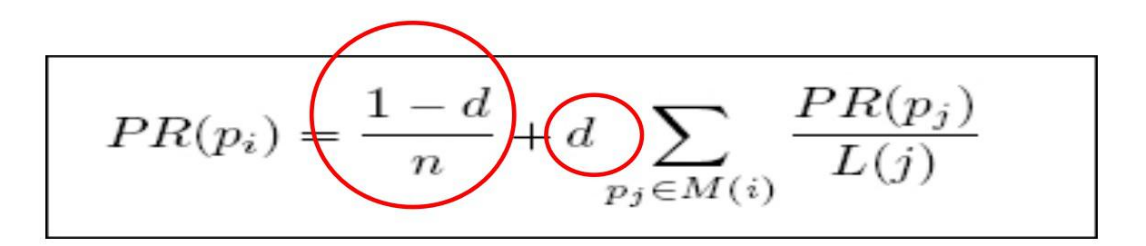
概率转移矩阵，在 R 的示例代码，如下。

```
Matrix04<-function(G,d=0.85){
cs<- colSums(G)
cs[cs==0] <- 1
n <- nrow(G)
delta<-(1-d)/n
# 这个 delta 是 5.1.2 没有考虑的，但是出现在 6.1.1.的公式里。

A<-matrix(delta,nrow(G),ncol(G))
for (i in 1:n) A[i,] <- A[i,] + d*G[i,]/cs
A}
# 这里的- A[i,] + d*G[i,]/cs 是完全按照图 6.7 的公式而来。

G<-Matrix04(A)
G
```

因为考虑了居尼系数，所以此处公式完全按照图 5.7 公式而来，我们也可借此感受到建立以数学公式为基础并且以 R 代码为实现的乐趣。图 5.14 说明我们做了图 5.7 的那个部分。



$$PR(p_i) = \frac{1-d}{n} + d \sum_{p_j \in M(i)} \frac{PR(p_j)}{L(j)}$$

图 5.15 居尼系数（R）

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> A<-Matrix01(pages)
> A
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    1    0    0    1
[3,]    1    1    0    0
[4,]    1    1    1    0
> Matrix04<-function(G,d=0.85){
+ cs<- colSums(G)
+ cs[cs==0] <- 1
+ n <- nrow(G)
+ delta<-(1-d)/n
+ A<-matrix(delta,nrow(G),ncol(G))
+ for (i in 1:n) A[i,] <- A[i,] + d*G[i,]/cs
+ A}
> G<-Matrix04(A)
> G
          [,1]   [,2]   [,3]   [,4]
[1,] 0.0375000 0.0375 0.0375 0.0375
[2,] 0.3208333 0.0375 0.0375 0.8875
[3,] 0.3208333 0.4625 0.0375 0.0375
[4,] 0.3208333 0.4625 0.8875 0.0375
>
>
```

图 5.16 考虑居尼系数后的概率转移矩阵（R）

如果比较一下图 5.13 和图 5.16 的概率转移矩阵，会发现有了 Delta 之后的概率转移矩阵，至少在直观上，数据显示地更为合理。

最后，计算 PageRank 值，其代码如前一样。

```
Matrix03<-function(G,iter=100){
iter<-10
n<-nrow(G)
x<-rep(1,n)
for (i in 1:iter) x<- G %*% x
x/sum(x)
}
q<-Matrix03(G,100)
q
```

结果如下：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> G<-Matrix04(A)
> G
          [,1]   [,2]   [,3]   [,4]
[1,] 0.0375000 0.0375 0.0375 0.0375
[2,] 0.3208333 0.0375 0.0375 0.8875
[3,] 0.3208333 0.4625 0.0375 0.0375
[4,] 0.3208333 0.4625 0.8875 0.0375
>
> Matrix03<-function(G,iter=100){
+ iter<-10
+ n<-nrow(G)
+ x<-rep(1,n)
+ for (i in 1:iter) x<- G %*% x
+ x/sum(x)
+ }
>
> q<-Matrix03(G,100)
> q
          [,1]
[1,] 0.0375000
[2,] 0.3738930
[3,] 0.2063759
[4,] 0.3822311
>
```

图 6.17 有了居尼系数后的 PageRank 值（R）

比较图 5.17 和图 5.14 我们可以发现：增加阻尼系数后，页面 1 具有 PR(1)=(1-d)/n=(1-0.85)/4=0.0375 数值。

这表示：无外链页面的最小值。

意思是，如果您按照 5.2.3.的方式进行计算，那么无外链接的页面，就没有任何价值（因为数值为 0），但是如果您按照本节 5.2.3.的方式计算，也就是增加了居尼系数，也就是完全按照 PageRank 的设想进行分析，那么任何页面都有价值，即使它是无外链接的页面，也有存在价值（数值为 0.375）。这么看来，确实比较合理。

四、算法优化

如同前面第一部分的社会网络分析，在那个大段代码的第一部分是向量函数，类似于我们这里 R 以及内嵌的一些向量计算的函数，而第一部分的大段代码的第二部分是矩阵计算函数，是把数值能够依照矩阵的数据结构进行分析所做的函数。

此处，我们虽然已在前面第三部分实现了 PageRank 但是如果是大规模计算，那么就可以进一步进行算法优化，此处的优化是指节省计算资源进行的改进。

在 R 的示例代码，大部分都是与第三部分相同的。

```
# 导入数据
pages<-read.table(file="D:/pagerank.csv",header=FALSE,sep=",")
names(pages)<-c("src","dist")
pages
# 邻接矩阵
Matrix01<-function(pages){
n<-max(apply(pages,2,max))
A<-matrix(0,n,n)
```

```
for(i in 1:nrow(pages))A[pages[i,]$dist,pages[i,]$src]<-1
A
}
A<-Matrix01(pages)
A
# 概率转移矩阵
Matrix04<-function(G,d=0.85){
cs<- colSums(G)
cs[cs==0] <- 1
n <- nrow(G)
delta<-(1-d)/n
A<-matrix(delta,nrow(G),ncol(G))
for (i in 1:n) A[i,] <- A[i,] + d*G[i,]/cs
A}
G<-Matrix04(A)
G
```

这次考虑：不用迭代，从矩阵中计算特征值。

```
Matrix05<-function(G){
x<-Re(eigen(G)$vectors[,1])
x/sum(x)
}
q<-Matrix05(G)
q
```

结果：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
+ delta<-(1-d)/n
+ A<-matrix(delta,nrow(G),ncol(G))
+ for (i in 1:n) A[i,] <- A[i,] + d*G[i,]/cs
+ A}
> G<-Matrix04(A)
> G
          [,1]   [,2]   [,3]   [,4]
[1,] 0.0375000 0.0375 0.0375 0.0375
[2,] 0.3208333 0.0375 0.0375 0.8875
[3,] 0.3208333 0.4625 0.0375 0.0375
[4,] 0.3208333 0.4625 0.8875 0.0375
>
> Matrix05<-function(G){
+ x<-Re(eigen(G)$vectors[,1])
+ x/sum(x)
+ }
>
> q<-Matrix05(G)
> q
[1] 0.0375000 0.3732476 0.2067552 0.3824972
>
>
>
> 
```

图 5.18 利用矩阵进行最后的 PR 值计算（R）

比较一下图 5.17 和图 5.18 的区别，我们可以发现，其 PR 值都是相同的。然而，它们的计算方式，一个采取迭代计算，一个采取矩阵计算，效率上有所差异。

至于，如何看到差异，也就是看到 R 的计算资源消耗，或者从 R 或者 Python 管理计算机计算资源等等，需要等到第九章结束之后，再做讲解。

此处，可以通过本节第二部分内容去更好的理解 pageRank 的原理，然后，从本节第二部分和第三部分的比较，知道居尼系数在 PageRank 里的意义。接着，从第三部分和第四部分的比较，了解矩阵计算和迭代的差别，最后，尽管是 R 的 PageRank 的试验，但是能够让我们从第四部分去更好地理解第一部分里 Python 的 Social Network Analysis 的代码。

第三节 推荐

一、推荐算法

因为前面第二节已经把推荐算法的两大基础，社会网络分析和 PageRank 算法，进行介绍，所以此处就可进一步讨论推荐算法。

推荐算法本身也是一种排名方式，就是把第一名推荐出去，而且这种算法也是根据两两关系所建立的网络关系为基础。所不同的是，当我们说到“推荐”的时候，一般意义下，具有三个含义：

（一）根据数据进行数据分析；
（二）选用合适的算法；
（三）根据行为需求进行服务。

由此可见，我们并不是介绍一种算法，而是介绍一种服务，这种服务可能来自多种算法中的其中一种，又或者是多个算法的结果的比较之后，最合适的那个结果或者算法参数（第七章机器学习的重点）。

此处，我们仍然以 R 作为实现手段，具体实现步骤，如下：

（一）查看数据；
（二）建立数据模型；
（三）欧氏距离相似度算法；
（四）最近邻算法；
（五）基于用户的商品推荐算法；
（六）运行程序；
（七）观察结果；
（八）推荐用户。

在 R 的示例代码，如下。首先是查看数据，此处也可利用 6.2.2.的方式自建。

```
getwd()
setwd("D:/mydata")
dir(getwd())
#查看工作路径
#设置工作路径
#查看查看工作路径都有啥东西

LookLook<-read.table(file="CollaborativeFiltering.csv",sep=",")
LookLook
#看看那个名为 CollaborativeFiltering 的 csv 文件。
#其中，那个 V1 是用户 id，那个 V2 是物品 id，那个 V3 是用户评分。

names(LookLook)<-c("uid","iid","pref")
LookLook
#对变量进行命名，再看就比较清楚。
```

```
R Console
> getwd()
[1] "D:/mydata"
> setwd("D:/mydata")
> dir(getwd())
[1] "CollaborativeFiltering.csv" "pagerank.csv"
> LookLook<-read.table(file="CollaborativeFiltering.csv",sep=",")
> LookLook
   V1  V2  V3
1   1 101 5.0
2   1 102 3.0
3   1 103 2.5
4   2 101 2.0
5   2 102 2.5
6   2 103 5.0
7   2 104 2.0
8   3 101 2.5
9   3 104 4.0
10  3 105 4.5
11  3 107 5.0
12  4 101 5.0
13  4 103 3.0
14  4 104 4.5
15  4 106 4.0
```

#其中，那个V1是用户id，那个V2是物品id，那个V3是用户评分。

图 5.19 进行推荐算法的示例数据（R）

接着，是建立数据模型（Data Model），代码如下。

```
FileDataModel<-function(file){            #数据模型函数
  data<-read.csv(file,header=FALSE)       #读取 csv 文件到计算机内存
  names(data)<-c("uid","iid","pref")          #增加变量名称
  user <- unique(data$uid)            #计算用户数
  item <- unique(sort(data$iid))          #计算商品数
  uidx <- match(data$uid, user)
  iidx <- match(data$iid, item)
  M <- matrix(0, length(user), length(item))      # 定义数据矩阵
  i <- cbind(uidx, iidx, pref=data$pref)
    for(n in 1:nrow(i)){                  #给矩阵赋值
    M[i[n,][1],i[n,][2]]<-i[n,][3]
  }
    dimnames(M)[[2]]<-item
  M                        #返回矩阵数据
}
```

图 5.20 的 M 以及计算结果，要到本节最末代码跑完，才会产生，但是这块内容属于矩阵函数图，便于对照，所以放置于此。

M 意指：矩阵变换，通过 FileDataModel()函数，输出物品矩阵。

```
R Console

>
> Document<-"CollaborativeFiltering.csv"
> NEIGHBORHOOD_NUM<-2
> RECOMMENDER_NUM<-3
> M<-FileDataModel(Document)
> M
     101 102 103 104 105 106 107
[1,] 5.0 3.0 2.5 0.0 0.0   0   0
[2,] 2.0 2.5 5.0 2.0 0.0   0   0
[3,] 2.5 0.0 0.0 4.0 4.5   0   5
[4,] 5.0 0.0 3.0 4.5 0.0   4   0
[5,] 4.0 3.0 2.0 4.0 3.5   4   0
> S<-EuclideanDistanceSimilarity(M)
> S
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.0000000 0.6076560 0.2857143 1.0000000 1.0000000
[2,] 0.6076560 0.0000000 0.6532633 0.5568464 0.7761999
[3,] 0.2857143 0.6532633 0.0000000 0.5634581 1.0000000
[4,] 1.0000000 0.5568464 0.5634581 0.0000000 1.0000000
[5,] 1.0000000 0.7761999 1.0000000 1.0000000 0.0000000
> N<-NearestNUserNeighborhood(S,NEIGHBORHOOD_NUM)
> N
     [,1] [,2]
```

图 5.20 矩阵函数图（R）

建立【欧式距离相似度】的矩阵转换的函数

```
EuclideanDistanceSimilarity<-function(M){
  row<-nrow(M)
  s<-matrix(0, row, row)    #相似度矩阵
  for(z1 in 1:row){
    for(z2 in 1:row){
      if(z1<z2){
        num<-intersect(which(M[z1,]!=0),which(M[z2,]!=0))  # 可计算的列
        sum<-0
        for(z3 in num){
          sum<-sum+(M[z1,][z3]-M[z2,][z3])^2
        }
        s[z2,z1]<-length(num)/(1+sqrt(sum))
        }
    }
  }
  s[which(s> 1)]<- 1      # 对算法的阈值进行限制等于 if (s[z2,z1]>1) s[z2,z1]<-1 在 s[z2,z1]<-length(num)/(1+sqrt(sum))之后
  s[which(s< -1)]<- -1    # 对算法的阈值进行限制等于 if (s[z2,z1]<-1) s[z2,z1]<- -1 在 s[z2,z1]<-length(num)/(1+sqrt(sum))之后
  ts<-t(s)          #补全三角矩阵
  w<-which(upper.tri(ts))
  s[w]<-ts[w]
  s         #返回用户相似度矩阵
}
```

图 5.21 要到最末才能出现结果，但是搁置此处是因为它来自：欧式距离相似度。

此处 S 是指：矩阵变换，通过 EnclideanDistanceSimilarity()函数，进行欧式相似度算法后的输出。

```
R Console
[4,]     1     5
[5,]     1     3
>
> Document<-"CollaborativeFiltering.csv"
> NEIGHBORHOOD_NUM<-2
> RECOMMENDER_NUM<-3
> M<-FileDataModel(Document)
> M
     101 102 103 104 105 106 107
[1,] 5.0 3.0 2.5 0.0 0.0   0   0
[2,] 2.0 2.5 5.0 2.0 0.0   0   0
[3,] 2.5 0.0 0.0 4.0 4.5   0   5
[4,] 5.0 0.0 3.0 4.5 0.0   4   0
[5,] 4.0 3.0 2.0 4.0 3.5   4   0
> S<-EuclideanDistanceSimilarity(M)
> S
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.0000000 0.6076560 0.2857143 1.0000000 1.0000000
[2,] 0.6076560 0.0000000 0.6532633 0.5568464 0.7761999
[3,] 0.2857143 0.6532633 0.0000000 0.5634581 1.0000000
[4,] 1.0000000 0.5568464 0.5634581 0.0000000 1.0000000
[5,] 1.0000000 0.7761999 1.0000000 1.0000000 0.0000000
>
```

图 5.21 欧式距离相似度（R）

然后是：建立【最近邻】的矩阵转换的函数。

```
NearestNUserNeighborhood<-function(S,n){
  row<-nrow(S)
  neighbor<-matrix(0, row, n)
  for(z1 in 1:row){
    for(z2 in 1:n){
      m<-which.max(S[,z1])
      neighbor[z1,][z2]<-m
      S[,z1][m]=0
    }
  }
  neighbor                          # 返回前面 n 个最近邻
}
```

图 5.22 要到最末才能出现结果，但是搁置此处是因为它来自：最近邻。

此处 N 是指：通过用户相似度矩阵，通过 EuclideanDistanceSimilarity()函数，计算用户最近邻。

R Console

```
[1,] 5.0 3.0 2.5 0.0 0.0   0   0
[2,] 2.0 2.5 5.0 2.0 0.0   0   0
[3,] 2.5 0.0 0.0 4.0 4.5   0   5
[4,] 5.0 0.0 3.0 4.5 0.0   4   0
[5,] 4.0 3.0 2.0 4.0 3.5   4   0
> S<-EuclideanDistanceSimilarity(M)
> S
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.0000000 0.6076560 0.2857143 1.0000000 1.0000000
[2,] 0.6076560 0.0000000 0.6532633 0.5568464 0.7761999
[3,] 0.2857143 0.6532633 0.0000000 0.5634581 1.0000000
[4,] 1.0000000 0.5568464 0.5634581 0.0000000 1.0000000
[5,] 1.0000000 0.7761999 1.0000000 1.0000000 0.0000000
> N<-NearestNUserNeighborhood(S,NEIGHBORHOOD_NUM)
> N
     [,1] [,2]
[1,]    4    5
[2,]    5    3
[3,]    5    2
[4,]    1    5
[5,]    1    3
>
>
```

图 5.22 用户最近邻（R）

为了计算，建立【基于用户的推荐算法】的函数。

```
UserBasedRecommender<-function(uid,n,M,S,N){
  row<-ncol(N)
  col<-ncol(M)
  r<-matrix(0, row, col)
  N1<-N[uid,]
  for(z1 in 1:length(N1)){
    num<-intersect(which(M[uid,]==0),which(M[N1[z1],]!=0))      # 可计算的列
    for(z2 in num){
      r[z1,z2]=M[N1[z1],z2]*S[uid,N1[z1]]
    }
  }
  sum<-colSums(r)
  s2<-matrix(0, 2, col)
  for(z1 in 1:length(N1)){
    num<-intersect(which(colSums(r)!=0),which(M[N1[z1],]!=0))
    for(z2 in num){
      s2[1,][z2]<-s2[1,][z2]+S[uid,N1[z1]]
      s2[2,][z2]<-s2[2,][z2]+1
    }
  }
s2[,which(s2[2,]==1)]=10000
  s2<-s2[-2,]
  r2<-matrix(0, n, 2)
  rr<-sum/s2
  item <-dimnames(M)[[2]]
```

```
  for(z1 in 1:n){
    w<-which.max(rr)
    if(rr[w]>0.5){
      r2[z1,1]<-item[which.max(rr)]
      r2[z1,2]<-as.double(rr[w])
      rr[w]=0
    }
  }
  r2
}
```

第六步，运行程序，有了这步，才有图 5.20 图 5.21 图 5.22 的可能。

```
Document<-"CollaborativeFiltering.csv"      #数据文件
NEIGHBORHOOD_NUM<-2          # 取两个近邻
RECOMMENDER_NUM<-3           #保留三个推荐结果

M<-FileDataModel(Document)
# 把数据文件转换矩阵后，加载到内存。

S<-EuclideanDistanceSimilarity(M)
# 计算用户相似度矩阵

N<-NearestNUserNeighborhood(S,NEIGHBORHOOD_NUM)
# 计算用户近邻
```

第七步，是观察结果的示例代码，有了这步，才有图 5.20 图 5.21 图 5.22 的显示。

```
R1<-UserBasedRecommender(1,RECOMMENDER_NUM,M,S,N)
R1
#查看 User=1 的推荐结果

R2<-UserBasedRecommender(2,RECOMMENDER_NUM,M,S,N)
R2
#查看 User=2 的推荐结果

R3<-UserBasedRecommender(3,RECOMMENDER_NUM,M,S,N);R3
R4<-UserBasedRecommender(4,RECOMMENDER_NUM,M,S,N);R4
R5<-UserBasedRecommender(5,RECOMMENDER_NUM,M,S,N);R5
```

通过基于用户的推荐物品的 UserBasedRecommender()函数，把上述三个矩阵合并计算，得到每个用户的推荐物品和物品分数。如图 5.23 所示。

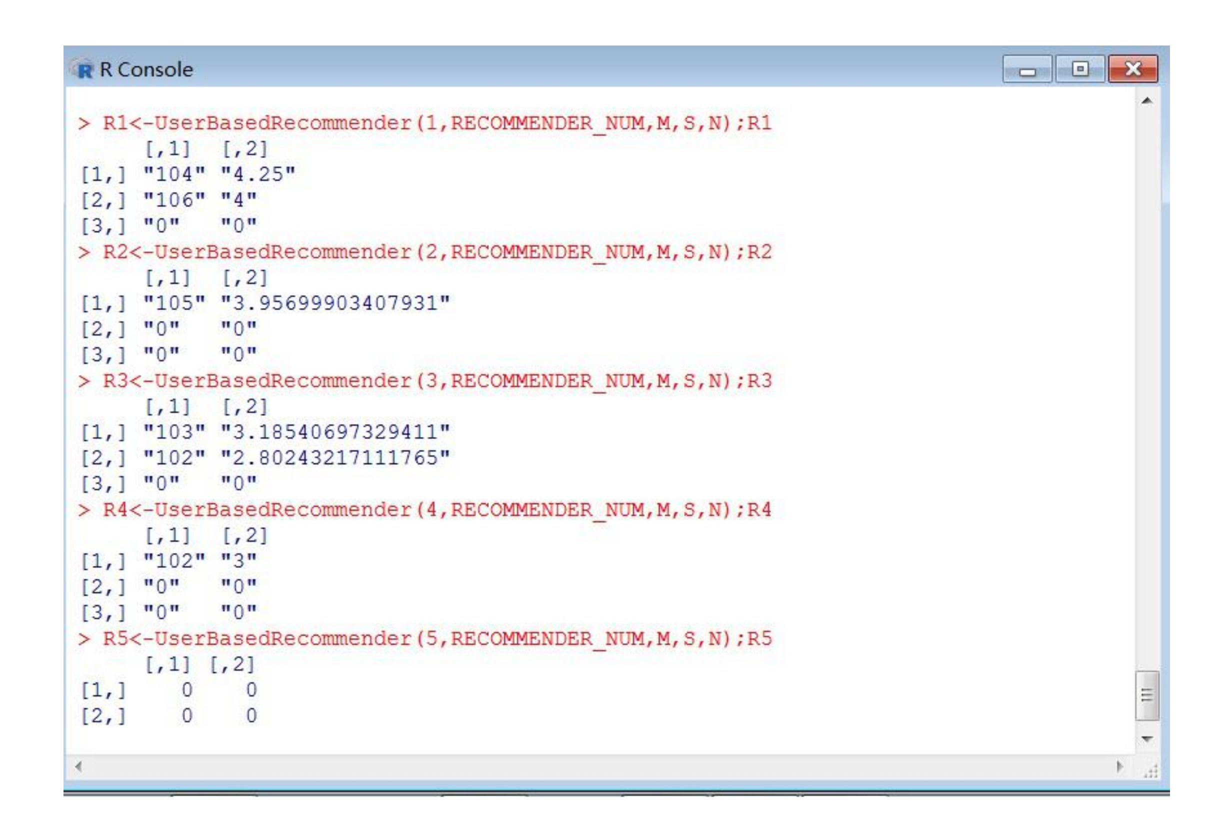


图 5.23 观察结果（R）

最后一步，是推荐物品，其代码如下。

```
#调整要求
#取前面两位分数最高的物品，进行用户推荐。
RECOMMENDER_NUM<-2

#向 1 号用户（uid=1）推荐物品以及告知得分。
R1<-UserBasedRecommender(1,RECOMMENDER_NUM,M,S,N)
R1
```

根据信息用户的数据：用户、产品、用户给产品的打分，这三组数据内容，进行信息推荐服务。结果显示：

```
R Console
[1,] "105" "3.95699903407931"
[2,] "0"   "0"
[3,] "0"   "0"
> R3<-UserBasedRecommender(3,RECOMMENDER_NUM,M,S,N);R3
     [,1]  [,2]
[1,] "103" "3.18540697329411"
[2,] "102" "2.80243217111765"
[3,] "0"   "0"
> R4<-UserBasedRecommender(4,RECOMMENDER_NUM,M,S,N);R4
     [,1]  [,2]
[1,] "102" "3"
[2,] "0"   "0"
[3,] "0"   "0"
> R5<-UserBasedRecommender(5,RECOMMENDER_NUM,M,S,N);R5
     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    0
> RECOMMENDER_NUM<-2
> R1<-UserBasedRecommender(1,RECOMMENDER_NUM,M,S,N)
> R1
     [,1]  [,2]
[1,] "104" "4.25"
[2,] "106" "4"
```

图 5.24 调整结果（R）

我们首先以欧式距离相似度，或者其他方法，进行矩阵转换后，处理数据。然而，采用哪种矩阵转换、哪种机器学习算法、哪些推荐规则等，均是按照最终用户体验，也就是推荐的物品准不准确、需不需要、用户有没有采取后续行动等来判断，因此归根到底还是客户在先，而研发在后，是先确保有优质数据，再考虑何种算法以及算法改进。

二、分布计算

为了优化计算效率，并且我们提前了解第八章的内容，此处介绍分布式大数据计算的方式，虽然我们使用的是 R 以及我们自己设计的小数据，但是我们实现的是本章第二节第二部分和第三部分的 PageRank 的另一种优化。

实现步骤，包括五项：

（一）建立 Map 程序

（二）建立 Reduce 程序

（三）计算矩阵特征值

（四）运行矩阵计算

（五）求得迭代结果

首先，涉及 Map 的 R 代码，如下。

```
map <- function(S0, node = "a") {
  S <- apply(S0, 2, function(x) x/sum(x))
  if (node == "a")
    S[, 1:2] else S[, 3:4]
}
```

其次，涉及 Reduce 的 R 代码，如下：

```
reduce <- function(A, B, a = 0.85, niter = 100) {
  n <- nrow(A)
  q <- rep(1, n)
  Ga <- a * A + (1 - a)/n * (A[A != 1] = 1)
  Gb <- a * B + (1 - a)/n * (B[B != 1] = 1)
  for (i in 1:niter) {
    qa <- as.matrix(q[1:ncol(A)])
    qb <- as.matrix(q[(ncol(A) + 1):n])
    q <- Ga %*% qa + Gb %*% qb
  }
  as.vector(q/sum(q))
}
```

接着，我们要进行【矩阵特征值】的计算，在 R 的示例代码，如下。

```
# 3.计算
S0 <- t(matrix(c(0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0),nrow = 4))
 # 此处，我们自建一个矩阵，然而，我们也可以导入矩阵的数据集进行分析。
A <- map(S0, "a")
B <- map(S0, "b")
reduce(A, B)
```

显示结果如图 5.25：在建立 MapReduce 程序后，导入矩阵数据，进行初步分析，

```
R Console
> map <- function(S0, node = "a") {
+   S <- apply(S0, 2, function(x) x/sum(x))
+   if (node == "a")
+     S[, 1:2] else S[, 3:4]
+ }
> reduce <- function(A, B, a = 0.85, niter = 100) {
+   n <- nrow(A)
+   q <- rep(1, n)
+   Ga <- a * A + (1 - a)/n * (A[A != 1] = 1)
+   Gb <- a * B + (1 - a)/n * (B[B != 1] = 1)
+   for (i in 1:niter) {
+     qa <- as.matrix(q[1:ncol(A)])
+     qb <- as.matrix(q[(ncol(A) + 1):n])
+     q <- Ga %*% qa + Gb %*% qb
+   }
+   as.vector(q/sum(q))
+ }
> S0 <- t(matrix(c(0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0),nrow = $
> A <- map(S0, "a")
> B <- map(S0, "b")
> reduce(A, B)
[1] 0.0375000 0.3732476 0.2067552 0.3824972
>
```

图 5.25 建立 MapReduce 程序（R）

实际上，我们最关心的是 PageRank 通过矩阵分析，并且是以分布式大数据计算的方式进行。在 R 的示例代码，如下。

```
reduce <- function(A, B, a = 0.85, niter = 100) {
```

```
  n <- nrow(A)
  q <- rep(1, n)
  Ga <- a * A + (1 - a)/n * (A[A != 1] = 1)
  Gb <- a * B + (1 - a)/n * (B[B != 1] = 1)
  for (i in 1:niter) {
    qa <- as.matrix(q[1:ncol(A)])
    qb <- as.matrix(q[(ncol(A) + 1):n])
    q <- Ga %*% qa + Gb %*% qb
  }
  print(q)
  print(as.vector(q/sum(q)))
}
```

运行结果，如下：

```
R Console
> reduce <- function(A, B, a = 0.85, niter = 100) {
+   n <- nrow(A)
+   q <- rep(1, n)
+   Ga <- a * A + (1 - a)/n * (A[A != 1] = 1)
+   Gb <- a * B + (1 - a)/n * (B[B != 1] = 1)
+   for (i in 1:niter) {
+     qa <- as.matrix(q[1:ncol(A)])
+     qb <- as.matrix(q[(ncol(A) + 1):n])
+     q <- Ga %*% qa + Gb %*% qb
+   }
+   print(q)
+   print(as.vector(q/sum(q)))
+ }
> reduce(A, B, niter=1)
          [,1]
[1,] 0.1500000
[2,] 1.2833333
[3,] 0.8583333
[4,] 1.7083333
[1] 0.0375000 0.3208333 0.2145833 0.4270833
> reduce(A, B, niter=2)
          [,1]
[1,] 0.1500000
```

图 5.26 进行矩阵计算（R）

最后，我们要求得迭代结果，在 R 的示例代码，如下。

```
reduce(A, B, niter=1)
reduce(A, B, niter=2)
reduce(A, B, niter=20)
```

如图 5.27 所示：

R Console

```
> reduce(A, B, niter=1)
          [,1]
[1,] 0.1500000
[2,] 1.2833333
[3,] 0.8583333
[4,] 1.7083333
[1] 0.0375000 0.3208333 0.2145833 0.4270833
> reduce(A, B, niter=2)
          [,1]
[1,] 0.1500000
[2,] 1.6445833
[3,] 0.7379167
[4,] 1.4675000
[1] 0.0375000 0.4111458 0.1844792 0.3668750
> reduce(A, B, niter=10)
          [,1]
[1,] 0.1500000
[2,] 1.4955721
[3,] 0.8255034
[4,] 1.5289245
[1] 0.0375000 0.3738930 0.2063759 0.3822311
>
>
```

图 5.27 推荐结果（R）

首先，这个案例，迭代到 10 次之后，就形成稳定，如果我们继续迭代到 20 次，其结果也是如图 5.27 所示的 miter=10 的结果一样。

其次，本节与第二节第四部分都是采取矩阵计算的方式实现 PageRank 的数值，所不同的是，采取了 MapReduce 的方式，因为在实际过程中，我们往往不是一组固定的数据集，而是多个样本不断流入，需要考虑到每次新的样本，都会改变网络关系，因此需要长期迭代。

然而，并不能采取前面第二节第三部分的迭代作法，因为，所以，将第二节三、四部分的优点合并，再以本节第一部分的编程方式予以实现。

三、词频计算

接下来，我们想要利用本节第一部分的推荐算法里的公式以及第二部分的 MapReduce 采取矩阵和迭代计算的方法，计算一组文本数据，把文本数据拆解为一组组的词语，根据词语之间的关系，进行信息推荐。

如此说来，我们需要考虑的第一件事，就是文本变成词频。将它所需要的函数，定义清楚之后，我们再加入刚刚所学的 MapReduce 就能形成所需要的工具集。接着，导入数据。之后，采取矩阵计算的方式，最终运行程序得到结果。

根据以上思路，我们所需要的工作环境，又迁移到 Python 的话，那么，整个工作流程的步骤，可以拆解为以下五个部分：

（一）建立工作环境；

（二）设置所需的类；

（三）进行分析状态的更新；

（四）矩阵乘法；

（五）执行程序。

首先，我们要能具有文本处理和数值计算的 Python 工作环境。

```
import math, random, re, datetime
from collections import defaultdict, Counter
from functools import partial
```

其次，也是最为重要的组成部分，就是我们想要进行词频计算所需的函数，代码如下：

```
# 2.设置所需的类
# 2.1.【标记化讯息】
def tokenize(message):
    message = message.lower()
    all_words = re.findall("[a-z0-9']+", message)
    # 转换小写
    # 抽取字词，从 a 到 z 从 0 到 9 的所有词
    return set(all_words)
    # 删除重复项

# 2.2.【字词计数】：还未用到 MapReduce 进行字词计算
def word_count_old(documents):
    # 返回那些来自文档中的标记化讯息中的字词的计数数量。
    return Counter(word
        for document in documents
        for word in tokenize(document))

# 2.3.【字词映射（mapper）】
def wc_mapper(document):
    # 对文档中的每个单词进行标记化（tokenize）。
    for word in tokenize(document):
        yield (word, 1)

# 2.4.【字词计算的简化器（reducer）】
def wc_reducer(word, counts):
    #对一个字词的计数的加总
    yield (word, sum(counts))

# 2.5.【字词计算的计算器】
def word_count(documents):
    # collect 是存储分组数值的位置
    collector = defaultdict(list)
    # 使用 MapReduce 计算输入文档的字词。
    for document in documents:
        for word, count in wc_mapper(document):
            collector[word].append(count)
    return [output
            for word, counts in collector.items()
            for output in wc_reducer(word, counts)]

# 2.6.【MapReduce】
def map_reduce(inputs, mapper, reducer):
```

```
    #输入，使用 mapper 和 reducer 运行 MapReduce
    collector = defaultdict(list)
    for input in inputs:
        for key, value in mapper(input):
            collector[key].append(value)
    return [output
            for key, values in collector.items()
            for output in reducer(key,values)]

# 2.7.【聚合键值对】
def reduce_with(aggregation_fn, key, values):
    # 通过 aggregation_fn 减少【键值（key-values）对】。
    yield (key, aggregation_fn(values))

# 2.8.【数值简化器】
def values_reducer(aggregation_fn):
    #把函数转换为 reducer(values -> output)。
    return partial(reduce_with, aggregation_fn)

# 2.9【计算式】：总和、最大值、最小值、距离
sum_reducer = values_reducer(sum)
max_reducer = values_reducer(max)
min_reducer = values_reducer(min)
count_distinct_reducer = values_reducer(lambda values: len(set(values)))
```

接着，我们导入数据，此处，我们假定有四位同学说了几段话语。根据这些话语，以及这些同学的基本信息（其实没有用到），我们进行后续的分析。

```
# 3.进行分析状态的更新
# 3.1.【状态更新记录】
status_updates = [
    {"id": 1,
     "username" : "Alan Ku",
     "text" : "Is anyone interested in a data science book ?",
     "created_at" : datetime.datetime(2019, 12, 14, 11, 37, 0),
     "liked_by" : ["Irina Wang", "Joe Liu", "George He"] },
     {"id": 2,
     "username" : "Irina Wang",
     "text" : "Which is the best book for learning data science ?",
     "created_at" : datetime.datetime(2020, 2, 2, 12, 24, 0),
     "liked_by" : ["Joe Liu", "Alan Ku", "Lisa Huang"] },
      {"id": 3,
     "username" : "Joe Liu",
     "text" : "How many classmate in your course ?",
     "created_at" : datetime.datetime(2020, 3, 3, 8, 30, 0),
     "liked_by" : ["Alan Ku", "Iina Wang", "Lisa Huang"] },
       {"id": 4,
     "username" : "Lisa Huang",
     "text" : "Do you like the course of practical data science with R and Python ?",
     "created_at" : datetime.datetime(2020, 5, 21, 11, 20, 0),
```

```
    "liked_by" : ["George He", "Joe Liu", "Alan Ku"] },]
```

我们还要把建立【词语映射】到【矩阵】的关系。此处，用到刚刚建立的【类】，也就是词语如何进行计数，计数结果的数值，在以下代码之中要变成矩阵所需要的数值。示例代码，如下。

```
# 3.2.【状态更新映射器】
def data_science_day_mapper(status_update):
    # 如果状态更新记录(status_update)包括"data science"的字词在内,则 yieldyields (day_of_week, 1) 数值为一。
    if "data science" in status_update["text"].lower():
        day_of_week = status_update["created_at"].weekday()
        yield (day_of_week, 1)

data_science = map_reduce(status_updates,
                              data_science_day_mapper,
                              sum_reducer)

# 3.3.【字词映射器】：根据每位用户映射字词
def words_per_user_mapper(status_update):
    user = status_update["username"]
    for word in tokenize(status_update["text"]):
        yield (user, (word, 1))

# 3.4.【最高字词计数的序列对】
def most_popular_word_reducer(user, words_and_counts):
    # 给定一个序列对（字词，计数），返回总计数最高的字词。
    word_counts = Counter()
    for word, count in words_and_counts:
        word_counts[word] += count
    word, count = word_counts.most_common(1)[0]
    yield (user, (word, count))
user_words = map_reduce(status_updates,
                        words_per_user_mapper,
                        most_popular_word_reducer)

# 3.5.【偏好映射器】
def liker_mapper(status_update):
    user = status_update["username"]
    for liker in status_update["liked_by"]:
        yield (user, liker)

distinct_likers_per_user = map_reduce(status_updates,
                                      liker_mapper,
                                      count_distinct_reducer)
```

以下，是矩阵计算的内容，包括两大部分，一是处理映射而来的数值，二是进行简化便于之后的计算。

```
# 4.矩阵乘法（matrix multiplication）
# 4.1.【矩阵相乘映射器】
def matrix_multiply_mapper(m, element):
    # 作为通用维度（A 列，B 行）的 m 元素，是一副元组 tuple（矩阵名称，i, j,值）。
    matrix, i, j, value = element

    if matrix == "A":
        for column in range(m):
            # 对于每个 C_i 列，总数的第 j 个实体是 A_ij。
            yield((i, column), (j, value))
    else:
        for row in range(m):
            # 对于每个 C_j 行，总数的第 j 个实体是 B_ij。
            yield((row, j), (i, value))

# 4.2.【矩阵相乘简化器】
def matrix_multiply_reducer(m, key, indexed_values):
    results_by_index = defaultdict(list)
    for index, value in indexed_values:
        results_by_index[index].append(value)
    # 总和结果个数为二的（len(result)==2）的所有值
    sum_product = sum(results[0] * results[1]
                      for results in results_by_index.values()
                      if len(results) == 2)
    if sum_product != 0.0:
        yield (key, sum_product)
```

执行程序的部分，便是调用之前定义好的函数，以及数据类型为矩阵的数据，经过处理呈现计算结果（含中文注释）而已。

```
# 5. 执行程序
if __name__ == "__main__":
    # 5.1.字词情况
    documents = ["data science", "big data", "data analysis", "big science"]

    wc_mapper_results = [result
                         for document in documents
                         for result in wc_mapper(document)]

    print("字词映射结果")
    print(wc_mapper_results)
    print()

    print("字词计数结果")
    print(word_count(documents))
    print()

    print("使用 MP 函数的字词计数结果")
    print(map_reduce(documents, wc_mapper, wc_reducer))
    print()
```

```
print("数据科学")
print(data_science)
print()

print("用户词组")
print(user_words)
print()

print("每位用户的不同爱好")
print(distinct_likers_per_user)
print()
print()

# 5.2.矩阵相乘（matrix multiplication）

entries = [("A", 0, 0, 3), ("A", 0, 1,  2),
       ("B", 0, 0, 4), ("B", 0, 1, -1), ("B", 1, 0, 10)]
mapper = partial(matrix_multiply_mapper, 3)
reducer = partial(matrix_multiply_reducer, 3)

print("矩阵相乘")
print("实体:", entries)
print()
print("结果:", map_reduce(entries, mapper, reducer))
print()
```

上述代码的运行结果，如图 5.28 所示。

```
字词映射结果
[('science', 1), ('data', 1), ('big', 1), ('data', 1), ('analysis', 1), ('data', 1), ('science', 1), ('big', 1)]

字词计数结果
[('science', 2), ('data', 3), ('big', 2), ('analysis', 1)]

使用MP函数的字词计数结果
[('science', 2), ('data', 3), ('big', 2), ('analysis', 1)]

数据科学
[(5, 1), (6, 1), (3, 1)]

用户词组
[('Alan Ku', ('science', 1)), ('Irina Wang', ('science', 1)), ('Joe Liu', ('classmate', 1)), ('Lisa Huang', ('science', 1))]

每位用户的不同爱好
[('Alan Ku', 3), ('Irina Wang', 3), ('Joe Liu', 3), ('Lisa Huang', 3)]


矩阵相乘
实体: [('A', 0, 0, 3), ('A', 0, 1, 2), ('B', 0, 0, 4), ('B', 0, 1, -1), ('B', 1, 0, 10)]

结果: [((0, 0), 32), ((0, 1), -3)]

>>>
```

图 5.28 词频计算结果（Python）

图 5.28 显示了所有文本当中出现哪些词语以及出现最高频率的哪些词语，也经过计算呈现了关键词语“数据科学”与其他词语的关系，以及各个用户喜好的词语及其“数据科学”有关的词语，最后形成新的矩阵的数据。

至此，我们通过原始数据（文本），形成了新的衍射数据（矩阵），并且具有一定的研究数据和结论（即使此处因为文本量少，结果并不突出和显眼）。

四、相似推荐

接续从本节第一部分和第二部分的 R 的推荐算法和分布式计算，以及第三部分的词频计算的基础，我们如果进行协调过滤的推荐，即，更为精确的计算和推荐的任务，可以合并已知的几种技术手段。

在第二节，我们为了理解，所以可以尽量详细，在本节的部分则是到此，我们又可【化繁为简】，把整项数据建模，化约为四个部分：

（一）工作环境与数据
（二）用户协同过滤
（三）物品协同过滤
（四）计算结果

以下，是上述各个部分的分段代码以及注释。

```
# 1.准备工作环境
import math, random
from collections import defaultdict, Counter

# 1.1.【两对象相乘相加】：公式为 v_1 * w_1 + ... + v_n * w_n 写成函数变成类，便于以下各处使用。
def dot(v, w):
    return sum(v_i * w_i for v_i, w_i in zip(v, w))

#1.2.【用户感兴趣的书名】：导入 0-14 共 15 组的词，每一组代表一位用户，每一组的词，代表该用户的兴趣/书名。
users_interests = [
    ["Spark", "Big Data", "HBase", "Java", "Hadoop", "Storm", "Cassandra"],
    ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
    ["Python", "scikit-learn", "machine learning", "numpy", "statistics", "MySQL"],
    ["R", "Python", "statistics", "regression", "probability"],
    ["machine learning", "regression", "decision trees", "KNN"],
    ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
    ["statistics", "probability", "mathematics", "theory"],
    ["machine learning", "scikit-learn", "Mahout", "neural networks"],
    ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
    ["Hadoop", "Java", "MapReduce", "Big Data"],
    ["statistics", "R", "Python"],
    ["statistics", "deep learning", "artificial intelligence", "probability"],
    ["MySQL", "R", "Python"],
    ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
    ["KNN", "regression", "support vector machines"]
]

# 1.3.【受欢迎的程度】
popular_interests = Counter(interest
                            for user_interests in users_interests
                            for interest in user_interests).most_common()
## 函数.most_common()用来统计出现的次数。从 user_interests 找到字符串组成的
```

```
interest（词名）及其词频。


# 1.4.【最受欢迎的新书名】
def most_popular_new_interests(user_interests, max_results=3): #
    ## 此处代入 1.2.【用户感兴趣的书名】以及设置输出多少结果（max_results）。
    suggestions = [(interest, frequency)
                   for interest, frequency in popular_interests
                   if interest not in user_interests]
    return suggestions[:max_results]
## 此处把 interest 和 frequency 的值，循环代入到 1.3.【受欢迎的程度】的类里。
## 返回的”建议“就是计算后的 interest（词名，字符串）和 frequency（词频，数值），个数为 max_results 所决定。
```

基于用户的协同过滤的 Python 代码如下

```
# 2.基于用户的过滤器
# 2.1.【余弦相似性】：返回 1.1.【两对象相乘相加】除以【两对象自我相乘再相乘的开平方】
def cosine_similarity(v, w):
    return dot(v, w) / math.sqrt(dot(v, v) * dot(w, w))

# 2.2.【唯一兴趣】：就是找到”词名“，此处，在 user_interests 中找到独立的字符串作为 interest 列表，并且把列表里的元素进行排序。
## 函数 sorted：对所有可迭代的对象进行排序操作。
unique_interests = sorted(list({ interest
                                 for user_interests in users_interests
                                 for interest in user_interests }))

# 2.3.【用户兴趣向量】：建立一个序列，以向量形式存储起来。
def make_user_interest_vector(user_interests):
    # 给定一副 interest 列表如果 unique_interests[i]在列表里，产生一个第 i 位的向量，否则为零。
    return [1 if interest in user_interests else 0
            for interest in unique_interests]

# 2.4.【用户兴趣矩阵】：映射 2.3.【用户兴趣向量】和 1.2.【用户感兴趣的书名】为一矩阵，但以列表形式存储起来。
user_interest_matrix = list(map(make_user_interest_vector, users_interests))

# 2.5.【用户相似性】 ：利用 2.1.【余弦相似性】计算出两两 2.3【用户兴趣向量】（i 和 j）的值
user_similarities = [[cosine_similarity(interest_vector_i, interest_vector_j)
                      for interest_vector_j in user_interest_matrix]
                     for interest_vector_i in user_interest_matrix]

# 2.6.【最相似的用户】：利用 2.5.【用户相似性】把每一个用户 id 代入，进行枚举，取得键值对（新建用户 id，用户相似性的值）
def most_similar_users_to(user_id):
    pairs = [(other_user_id, similarity)
```

```
    # 发现其他用户
             for other_user_id, similarity in
    # 与用户相关的其他用户（通过枚举法检查每个用户 id）
                enumerate(user_similarities[user_id])         # 非零
             if user_id != other_user_id and similarity > 0]  # 相似性

    return sorted(pairs,                                       # 把键值对进行排序
                  key=lambda pair: pair[1],                    # 找到最相似的作为第一位
                  reverse=True)                                # 表达式 lambda 定义了一个匿名函数，翻转键值对的值，并且返回到键值对里

# 2.7【基于用户的建议】
def user_based_suggestions(user_id, include_current_interests=False):
    # 相似性的值的加总
    suggestions = defaultdict(float)
    for other_user_id, similarity in most_similar_users_to(user_id):
        for interest in users_interests[other_user_id]:
            suggestions[interest] += similarity

    # 转换为一张经过排序后的列表。
    suggestions = sorted(suggestions.items(),
                         key=lambda pair: pair[1],
                         reverse=True)

    # 排除已有兴趣
    if include_current_interests:
        return suggestions
    else:
        return [(suggestion, weight)
                for suggestion, weight in suggestions
                if suggestion not in users_interests[user_id]]
```

基于商品的协同过滤（Item-Based Collaborative Filtering）的 Python 代码如下。

```
# 3.【基于商品的协同过滤（Item-Based Collaborative Filtering）】
# 3.1.【商品用户矩阵】
interest_user_matrix = [[user_interest_vector[j]
                         for user_interest_vector in user_interest_matrix]
                        for j, _ in enumerate(unique_interests)]
# 3.2.【商品相似性】
interest_similarities = [[cosine_similarity(user_vector_i, user_vector_j)
                          for user_vector_j in interest_user_matrix]
                         for user_vector_i in interest_user_matrix]

# 3.3.【用户共同感兴趣的商品】
def most_similar_interests_to(interest_id):
    similarities = interest_similarities[interest_id]
    pairs = [(unique_interests[other_interest_id], similarity)
```

```
            for other_interest_id, similarity in enumerate(similarities)
            if interest_id != other_interest_id and similarity > 0]
    return sorted(pairs,
                  key=lambda pair: pair[1],
                  reverse=True)

# 3.4.【基于商品的建议】
def item_based_suggestions(user_id, include_current_interests=False):
    suggestions = defaultdict(float)
    user_interest_vector = user_interest_matrix[user_id]
    for interest_id, is_interested in enumerate(user_interest_vector):
        if is_interested == 1:
            similar_interests = most_similar_interests_to(interest_id)
            for interest, similarity in similar_interests:
                suggestions[interest] += similarity

    suggestions = sorted(suggestions.items(),
                         key=lambda pair: pair[1],
                         reverse=True)

    if include_current_interests:
        return suggestions
    else:
        return [(suggestion, weight)
                for suggestion, weight in suggestions
                if suggestion not in users_interests[user_id]]
```

最后，实现结果的代码如下：

```
# 4.【执行图书推荐】

if __name__ == "__main__":

    print("【把书名按照受欢迎程度排序如下】：")
    print(popular_interests)
    print()
 ## 此处，第一个 print 是为了显示双引号内的中文字符，真正重要的是第二个 print 显示出计算结果，第三个 print 是为了保持间隔，纯粹美观而已，也可以用 /n 的正则表达式。
 ## 以下代码，情况雷同。
    print("【最受欢迎的新书名：】")
    print("与这些关键词 NoSQL、MongoDB、Cassandra、HBase、Postgres 相关的书：")
    print(most_popular_new_interests(["NoSQL", "MongoDB", "Cassandra", "HBase",
"Postgres"]))
    print()
    print("与这些关键词 R、Python、statistics、regression、probability 相关的书：
")
    print(most_popular_new_interests(["R", "Python", "statistics", "regression",
"probability"]))
    print()
```

```
print("【基于用户相似度的推荐】")
print("与第 1 号用户具有相同兴趣的用户：")
print(most_similar_users_to(0))

print("根据这些人的阅读书单推荐给第 1 号用户的书名：")
print(user_based_suggestions(0))
print()

print("【基于商品相似度的推荐】")
print("与'Big Data'最接近的书名，应该包括：")
print(most_similar_interests_to(0))
print()

print("推荐给第 1 号用户的书名，应该包括：")
print(item_based_suggestions(0))
```

结果：



图 5.29 图书推荐结果（Python）

如图 5.29 所示，我们把物品做了排序，把人推荐给人，把无推荐给人。

本章小结

本章介绍的数据建模,首先较长篇幅在于说明数据建模,接着简短介绍一些数据统计(延续第四章)和机器学习(下接第七章)的选型原则;但是,较大篇幅放在以社会网络、网页排名、信息推荐、分布式计算等的实践操作,这样,便于在一方面从动手操作的过程里理解本章内容,另一方面,强化和补充R与Python的基本技能。本章主要讲述了以下内容:

(1)数据建模的概念、选型、流程等,主要概念是:问题驱动建模,而不是算法在先或者模型在先。其他包括;根据数值变量和因子变量等,适合选取的建模方式,以及一般情况下的建模工作流程。

(2)社会网络分析;此处我们介绍了Python的类和面向对象编程的技法,把5.3.5的部分进一步深化为可实现的代码和程序。

(3)网页排名的基本建模;此处我们介绍了从PageRank应用于论文和专利等的方式,连带复习了4.3.1和4.3.2的内容。

(4)信息推荐的基本建模:此处我们除了介绍了小数据的推荐和大数据的推荐,在实现方案上的不同,先为第8章建立初步理解的基础。

(5)掌握分布式计算的基本建模;此处我们介绍两种不同的方式,一是就问题和算法进行建模的试验阶段,二是就问题和解决方案进行的建模的测试阶段。复习了1.1.1.的两种思维下的不同认识。

最后,进一步能够阅读R与Python代码,以及思考如何应用在数据科学上。

习　　题

一、单选题

1、在R语言中,哪个语句可以正确显示出today is a little cold这句话:(___)

(A)sprintf(“today is %s", “a little cold”)

(B)print “today is a little cold”

(C)type “today is a little cold”

(D)print(today is a little)

2、Python的内置函数【表述错误】的是:(___)

(A)abs() 求绝对值

(B)pow() 分别取商和余数

(C)round() 四舍五入

(D)storted()队集合排序

二、多选题

1、进行初步建模的顺序是(___)

(1)模型

(2)数据

(3)问题

（4）检验

2、利用已知算法解决问题的顺序是（___）
（1）算法优化
（2）算法实现
（3）算法描述
（4）解题步骤

三、问答题
1、请简要说明网页排序的原理以及举例说明怎么应用？
6、请说明数据建模的概念、选型、流程怎么应用？

四、课后作业

如果您完成第二、三、四、五章的课后作业，那么第六章的作业，分为两个步奏。

（1）为 10 篇论文，设定编号。基于第三章的作业，这里增加一个编号。此步奏与上周作业先沟通那个。

（2）为 10 篇论文，解读：

是否可以得到数据？Y/N

如可，给出网址，如不可，一句话说明理由。

五、作业答疑

问题：第二、三、四、五章的课后作业，是一张表格么？

回答：每次所做的作业，都是不同的.csv 文件。但是都是针对第一周作业的 10 篇研究论文。并且给于相同的编号。所以，这次作业，有三列：

（1）序号

（2）Y 或者 N

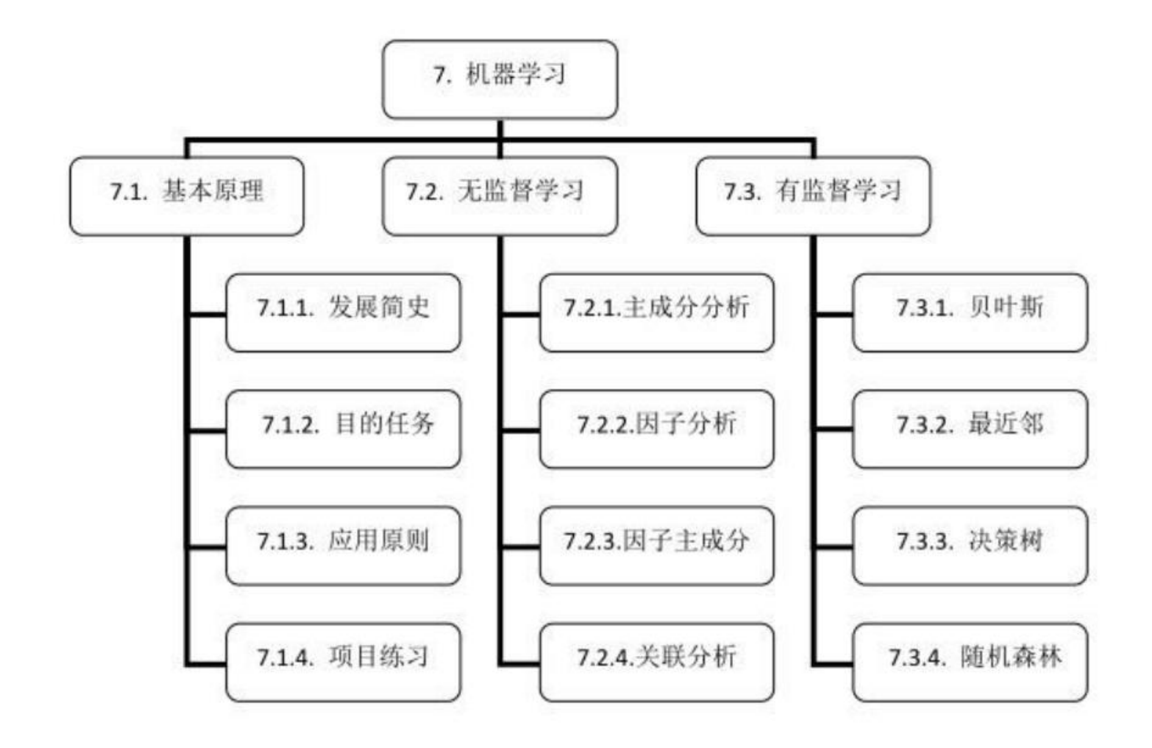
（3）网址 或者 一句话得不到论文里的数据的理由。

第六章　机器学习篇

学习目标

理解机器学习的基本原理；
理解无监督学习和有监督学习；
掌握主成分分析、因子分析、关联分析等；
掌握因子分析与主成分分析的综合用法；
掌握贝叶斯、最近邻、决策树等；
掌握随机森林与决策树的综合用法。

知识结构




导入案例

本章节，我们使用 R 进行学习，并且把一些已经做好的数据集，放到我们指定的路径里（您也可以设置成自己习惯找到的路径）。

第一节 基本原理

一、发展简史

机器学习是计算机算法的研究，通过经验自动改进。它的应用范围非常广泛，从发现大型数据集中一般规则的数据挖掘程序，到自动了解用户兴趣的信息过滤系统。机器学习有两个定义：

（一）发明计算机算法、把数据转化为智能行为。

（二）核心：找出复杂数据的意义。

机器学习是统计学、数据库科学、计算机科学的交叉学科。与机器学习密切相关的两组概念：

（一）数据挖掘。作为一门数据应用科学，包括：数据集成、数据清洗、统计建模、机器学习、可视分析等。

（二）人工智能。按照“图灵测试”的标准，涉及：知识表达、自然语言理解、机器学习、机器感知等。

机器学习虽然是个强大工具，能够从海量数据中找到真知灼见，但仍需要遵循科学方法与原理开展研究，保持谨慎态度，避免“拿着锤子找钉子”式地滥用。

机器学习是以输入数据以及学习任务为基础进行算法选择的一种研究方法。它的演进过程就是它的发展历史。

表 6.1 机器学习发展历史

年代	特征	重要成果
1950s	神经科学的理论基础	1.James 关于神经元是相互连接的发现 2.McCullon & Pitts 的神经元模型 3.Hebb 学习律（相互连接强弱度的变换规则）
1960s	感知器（Perceptron）时代	1.1957 年 Rosenblatt 首次提出； 2.1969 年：《Perceptron》出版，提出著名的 XOR 问题。
1970s	符号主义、逻辑推理	1.形式逻辑兴盛引导新的解题和提问。 2.更多纯数学、物理理论、数值计算的引入。
1960s ~1970s	统计学习理论创立	1.VC 维的基本概念； 2.结构风险最小化原则； 3.概率空间的大数定律。
1980s	连接主义	1.MLP+BP 算法成功解决 XOR 问题； 2.进入神经网络时代（但是受限于计算能力）。
1990s	统计学习理论的发展及完善	1.典型代表：SVM （Vapnik，Bell 实验室）； 2.结构风险最小化； 3.最小描述长度原则； 4.小样本问题； 5.核函数、核空间变化； 6.PAC 理论下的弱可学习理论的建立； 7.支持向量机 SVM 的充分应用。
2000s	各种机器学习理论及算法得以充分发展	1.符号机器学习 2.计算机器学习（统计学习理论，典型例子：SVM） 3.集群机器学习（典型代表：Boosting）

		4.强化机器学习 5.流行机器学习 6.监督学习，非监督学习 7.半监督学习、
2010s	深度学习开始快速发展	

机器学习是一个多学科交叉的产物，当前的发展现况是它吸取了人工智能、概率统计、神经生物学、认知科学、信息论、控制论、计算复杂性理论、哲学等学科的成果。

机器学习实际上是一个应用驱动的学科，其根本驱动力是：“更多、更好地解决实际问题”。由于近 20 年的飞速发展，机器学习已经具备了一定的解决实际问题的能力，似乎逐渐开始成为一种基础性、透明化的“支持技术、服务技术”。

（一）基础性：在众多的学科领域都得以应用；

（二）透明化：用户看不见机器学习，看见的是防火墙、生物信息、搜索引擎。

机器学习的广泛应用。机器学习在很多科技应用领域发挥了重要的实用价值，特别是在数据挖掘、语音识别、图像处理、机器人、车辆自动驾驶、生物信息学、信息安全、遥感信息处理、计算金融学、工业过程控制等。

机器学习在许多社会生活应用也发挥重要作用，例如，识别并且过滤垃圾邮件、预测犯罪活动、根据路况实现交通信号灯的自动化、给出暴风雨和自然灾害后经济损失的估计、检测客户流失、评估个人捐助能力、把广告定位到特定类型的客户等。

机器学习的目的：为发现或者预测。如果机器能够获取经验并且能利用它们，在以后的类似经验中能够提高他的表现，就称之为机器学习。



图 6.1 发现规律和预测未来

图 6.2 机器学习的应用

二、目的任务

机器学习模型的主要目的，是实现从【训练空间】（数据集）降维到【解决空间】（模

型及其参数）的过程。这个过程，包括许多机器学习的任务，可以简单叙述如下：
【训练数据空间→机器学习→解决空间（发现）或者算子（预测）】

训练数据空间，即：我们把数据集经过处理和规整之后，进行初步的数据统计，以及最终形成可供运算的矩阵、数据框等对象。

机器学习，即：在各种算法之下，从领域知识、前人研究成果、各种算法的试验结果等方式，找到合适的建模方式，然后开始进一步求取参数、提升性能、探索优化。

解决空间，即：通过机器学习发现隐藏在数据集之后的规律或者现象，或者，通过机器学习形成能够预测新的数据的【算子】等的任务。

表 6.2 机器学习的任务流程

流程	任务
1.分析研究	数据取样； 初始特征标记； 建立简单模型进行评估；
2.特征工程	数据训练； 改进特征；
3.模型评估	尝试不同的算法； 比较不同的结果。

机器学习是利用大量的数据样本，使得计算机通过不断的学习获得一个模型，用来对新的未知数据做预测。

（一）有监督学习（分类、回归）：同时将数据样本和标签输入给模型，模型学习到数据和标签的映射关系，从而对新数据进行预测。

（二）无监督学习（聚类）：只有数据，没有标签，模型通过总结规律，从数据中挖掘出信息。

（三）强化学习：在没有任何标签的情况下，通过先尝试做出一些行为得到一个结果，通过这个结果是对还是错的反馈，调整之前的行为，就这样不断的调整，算法能够学习到在什么样的情况下选择什么样的行为可以得到最好的结果。

一般而言，机器学习的范围主要是有监督学习和无监督学习两类，第三类的强化学习部分，我们在第九章深度学习的时候，再做讨论。

所以，我们如果想要了解机器学习，就先理解【聚类】与【分类】。

聚类 Clustering：对无类别标识的数据进行机器学习，发现其同类规律。

（一）训练空间：数据集

（二）维度：数据属性

（三）解决空间：聚类结果集

（四）维度+1：类别标签

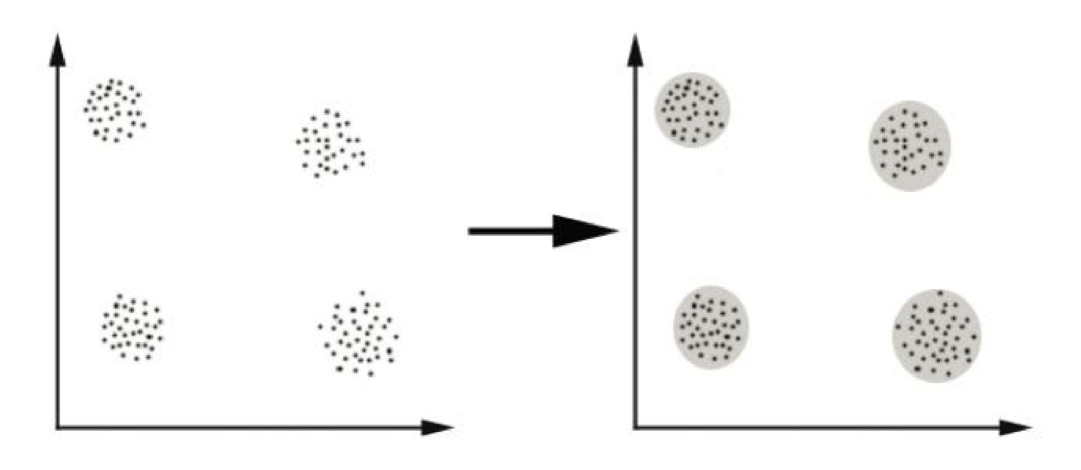


图 6.3 聚类示意图

聚类 Clustering

Step 1，随机选取 k 个点作为聚类种子。

Step 2，计算每个训练数据到种子点的距离，并归属为最近的类别。

Step 3，对每个类别重新计算种子中心。

Step 4，将所有训练点对新的种子距离运算，划分新类别。

Step 5，重复前两个步骤，直到类别稳定。

聚类 K-means

（一）探索数据：建特征属性。

（二）探索数据：给出 K 值。

（三）运算过程：距离计算。

（四）运算过程：初始种子选取。

（五）学习结束：收敛原则。

（六）学习结束：打标签。

分类（KNN 最近邻）：采用已分类数据集，对未知分类数据点进行类别标记。

（一）训练空间：已分类数据集，未知分类数据点；

（二）解决空间：未知分类数据点+分类标记；

（三）最近 k 个点中所属点最多的分类。

图 6.4 分类示意图

分类（KNN 最近邻）

（一）初始化距离为最大值

（二）计算未知样本和每个训练样本的距离 dist

（三）得到目前 K 个最临近样本中的最大距离 maxdist

（四）如果 dist 小于，则将该训练样本作为 K-最近邻样本

（五）重复步骤（二）、（三）、（四）maxdist 直到所有训练样本的距离算完

（六）统计 K-最近邻样本中每个类标号出现的次数

（七）选择出现频率最大的类标号作为未知样本的类标号。

机器学习的类型

机器学习的任务是：给定（训练）数据，通过算法，发现一些潜在的模式并将这个模式应用于新数据。

（一）监督学习 Supervised Learning

（二）非监督学习 Unsupervised Learning

（三）半监督学习 Semi-supervised Learning

监督学习：用于训练的数据有标记

（一）分类 Classification（学习决策边界），例如：文本 / 图片 / 视频分类，垃圾邮件，客户忠诚度预测等。

（二）回归 Regression（学习预测连续值）， 例如：预测房价，预测用户愿意支出的金额等。

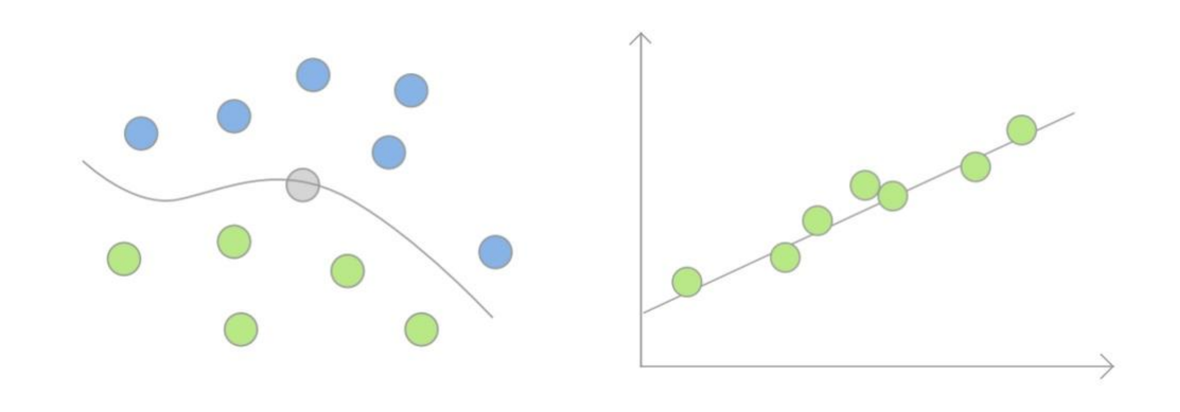


图 6.5 监督学习示意图

非监督机器学习：输入数据没有标记，寻找隐藏结构。

（一）聚类（Clustering），例如：对客户进行聚类画像，分析客户不同喜好。

（二）异常检测（Outlier /anomaly detection），例如：保险骗保、金融风险等。

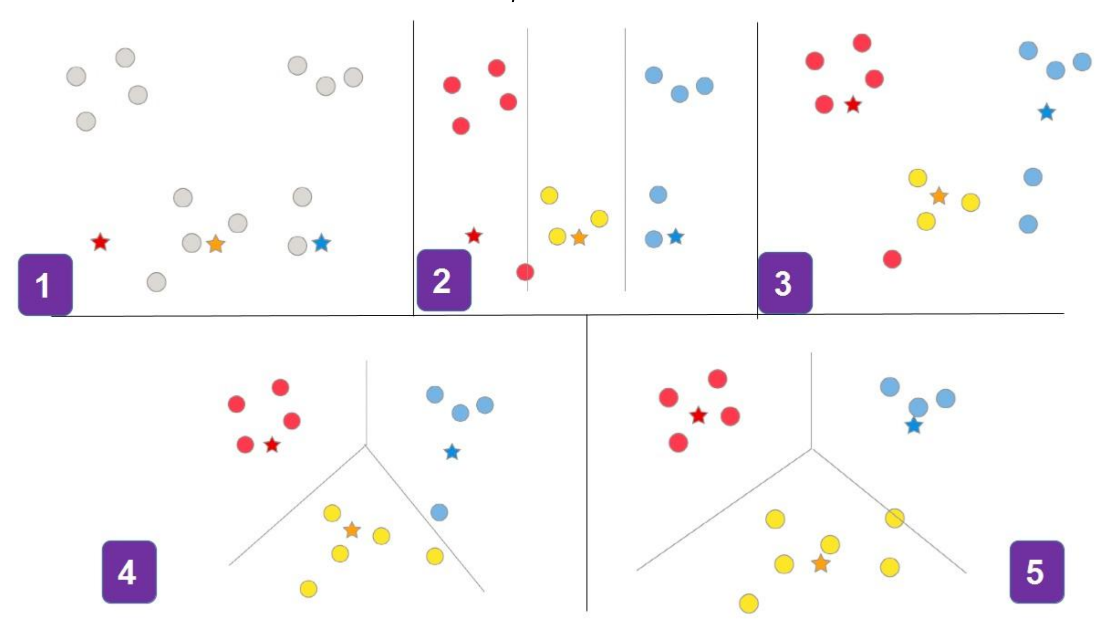


图 6.6 非监督学习示意图

机器学习的应用原则

奥卡姆剃刀：简单有效原理

（一）公元 14 世纪，英国萨里的奥卡姆（William of Ockham）对当时无休无止的关于“共相”、“本质”之类的争吵感到厌倦，于是著书立说，宣传只承认确实存在的东西，认为那些空洞无物的普遍性要领都是无用的累赘，应当被无情地“剃除”。他所主张的“思维经济原则”，概括起来就是“如无必要，勿增实体。”

（二）过多的考虑特征和数据，反而会影响机器学习的结果：

增加了无谓的计算开销；

影响属性的权重；

可能造成过度拟合。

（三）简言之，当有两个机器学习方法均能得出同样的结论时，那么简单的那个更好。

Bonferroni's Principle

（一）随着数据规模的增加，任何数据都会显现出一些不同寻常的特征，这些特征看上去非常重要，实际上却并不重要。

（二）在数据随机性假设的基础上，计算所寻找的事件的发生的期望值，如果该期望值大于找到的真实事件的数目，则所找到的事件是假象。

Phine Paradox

（一）Joseph Rhine 是 1950 年代的心理学家，他猜想某些人有超感知能力。他设计了一个实验：要求实验对象猜 10 张隐藏的卡片的颜色：红或者蓝？

（二）他发现 1000 个人里面有 1 个具有超感知能力：能猜对所有 10 张卡片的颜色!他告诉这些人他们有超能力，并要求他们再做一次同样的实验。结果，这些人都失去了他们的超能力。

（三）这个心理学家总结道：你不能告诉人们他们具有超能力，否则他们就会失去超能力。

马太效应

（一）凡有的，还要加倍给他叫他多余；没有的，连他所有的也要夺过来。

（二）类似还有：

帕累托法则

八二原则

幂律分布

蝴蝶效应

（一）上个世纪 70 年代，美国一个名叫洛伦兹的气象学家在解释空气系统理论时说，亚马逊雨林一只蝴蝶翅膀偶尔振动，也许两周后就会引起美国得克萨斯州的一场龙卷风。

（二）系统是开放的，我们无法分析我们没有的数据，而是对系统内数据微小的变化进行响应。

三、应用原则

机器学习项目的流程

定义问题；

Step 1: 问题是什么?

Step 2: 为什么这个问题需要解决?

Step 3: 我们如何解决这个问题。

准备数据；

Step 1: 数据选择

Step 2: 数据处理

Step 3: 数据转换

评估算法；

Step 1: 初步检测

Step 2: 参数设置

Step 3: 测试集和训练集

Step 4: 交叉验证

Step 5: 测试算法

改良结果；

Step 1: 算法改良

Step 2: 方法改良

Step 3: 抽取特征

呈现结果。

Step 1: 背景脉络

Step 2: 关键问题

Step 3: 解决方案

Step 4: 发现

Step 5: 限制

Step 6: 结论（步骤 1 到 3）

准备试验环境

```
# 了解已经安装哪些 R 包
library()

# 了解已经加载哪些 R 包
search()

# 卸载当前已经载入的 R 包
detach(package:已经载入的 R 包的正确名称)

# 了解已经安装哪些 R 包的细节
installed.packages()
## 名称、路径、版本、优先级别、依赖其他哪些辅助的 R 包、建议安装哪些辅助的 R 包、建议增强哪些 R 包、许可协议、是否许可、是否需要进行编译等。

# 如果只看名称、版本和路径
installed.packages()[,c("Package","LibPath","Version")]
```

#获取数据、初步了解数据。

```
nihao=read.csv("d:/mydata/iris.csv")
str(nihao)
install.packages("caret")
library(caret)
```

#获取数据、初步了解数据。

四、项目练习

研究动机与背景：

植物学家能够一眼识别“鸢尾花”的品种，但是专家的人力资源有限，如果采集一批数据后，能否采用“机器学习”的方式，自动进行判断。

问题描述：

已知“鸢尾花”的“特征”，如何判断“鸢尾花”的“品种”。

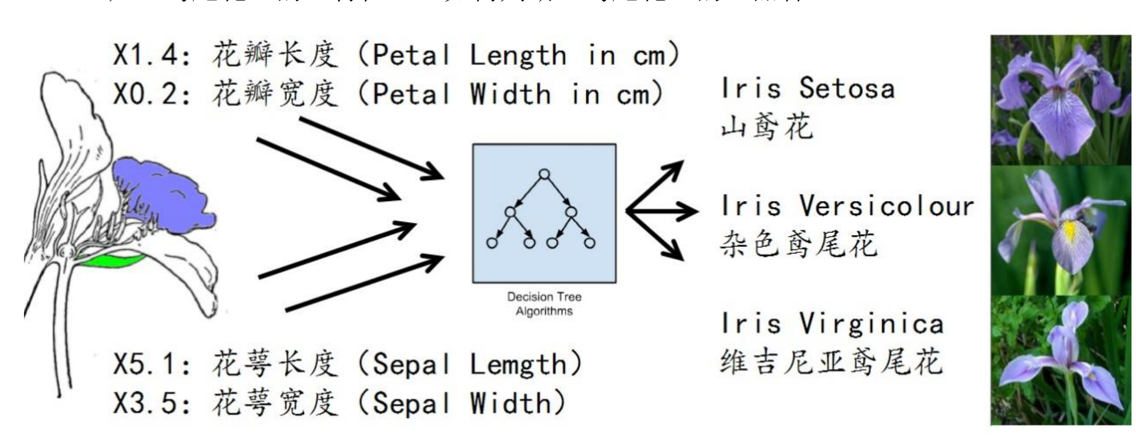


图 6.7 鸢尾花数据

准备数据

```
# 采用 iris 的数据集
nihao=read.csv("d:/mydata/iris.csv")

# 查看数据
str(nihao)
attributes(nihao)
```

结果：

```
R Console
> str(nihao)
'data.frame':   149 obs. of  5 variables:
 $ X5.1       : num  4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 5.4 ...
 $ X3.5       : num  3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 ...
 $ X1.4       : num  1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 ...
 $ X0.2       : num  0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 0.2 ...
 $ Iris.setosa: Factor w/ 3 levels "Iris-setosa",..: 1 1 1 1 1 1 1 1 1 1 ...
> attributes(nihao)
$names
[1] "X5.1"        "X3.5"        "X1.4"        "X0.2"        "Iris.setosa"

$class
[1] "data.frame"

$row.names
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
```

图 6.8. 查看鸢尾花数据集

准备数据

```
# 修改数据名称
```

```
colnames(nihao)<- c("Sepal-Length", "Sepal-Width", "Petal-Length", "Petal-Width", "Iris.setosa")

# 再次查看
attributes(nihao)
head(nihao)
tail(nihao)
```

结果：

```
R Console
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
[145] 145 146 147 148 149

> head(nihao)
  Sepal-Length Sepal-Width Petal-Length Petal-Width Iris.setosa
1          4.9         3.0          1.4         0.2 Iris-setosa
2          4.7         3.2          1.3         0.2 Iris-setosa
3          4.6         3.1          1.5         0.2 Iris-setosa
4          5.0         3.6          1.4         0.2 Iris-setosa
5          5.4         3.9          1.7         0.4 Iris-setosa
6          4.6         3.4          1.4         0.3 Iris-setosa
> tail(nihao)
    Sepal-Length Sepal-Width Petal-Length Petal-Width    Iris.setosa
144          6.7         3.3          5.7         2.5 Iris-virginica
145          6.7         3.0          5.2         2.3 Iris-virginica
146          6.3         2.5          5.0         1.9 Iris-virginica
147          6.5         3.0          5.2         2.0 Iris-virginica
148          6.2         3.4          5.4         2.3 Iris-virginica
149          5.9         3.0          5.1         1.8 Iris-virginica
>
```

图 6.9. 修改鸢尾花数据集的表头

评估算法

```
# 加载可以进行多个机器学习的 R 软件包
install.packages("caret")
library(caret)

# 设置训练集的控制函数
control <- trainControl(method="cv", number=10)
metric <- "Accuracy"

# 设置训练集的控制函数
control <- trainControl(method="cv", number=10)
## cv 意指：交叉验证（Cross-Validatory）,将会释放每个测试部分的所有组合，并且重复过程 3 次；当然“method”还有很多方法。number=10 意指：数据集分成 10 个部分，9 个部分用于训练，1 个用于测试。

# 设置“用来评估”的训练集的控制函数
metric <- "Accuracy"
```

```
##使用“精准度” 评估模型；在 caret 包，默认 RMSE 和 Rsquared 处理回归，而 Accuracy 和 Kappa 处理分类；若不设定，则系统报错；每个拆分得的比率，最后用以“正确预测实例数，除以总实例数，乘以 100%所得的百分数（比如精度为 95%）”的实际数量的比例通过。
```

评估四种不同算法

线性判别分析（LDA）

K 近邻（kNN）

支持向量机（SVM）

随机森林（RF）

```
# 进行线性判别分析（LDA）的建模
set.seed(7)
fit.lda <- train(Iris.setosa~., data=nihao, method="lda", metric=metric, trControl=control)
## 函数 train()用来进行基于训练集的建模。
## 加载 MASS 包，需要 e1070 包

# 进行 K 近邻（kNN）的建模
set.seed(7)
fit.knn <- train(Iris.setosa~., data=nihao, method="knn", metric=metric, trControl=control)
## 加载 rpart 包

# 进行支持向量机（SVM）的建模
set.seed(7)
fit.svm <- train(Iris.setosa~., data=nihao, method="svmRadial", metric=metric, trControl=control)
## 需要 kernlab 包。

#进行随机森林（RF）的建模
set.seed(7)
fit.rf <- train(Iris.setosa~., data=nihao, method="rf", metric=metric, trControl=control)
##加载 randomForest 包，就是之前上课用过的分类树的 R 包。

# 查看结果
RESULT <- resamples(list(lda=fit.lda, cart=fit.cart, knn=fit.knn, svm=fit.svm, rf=fit.rf))
summary(RESULT)
## 函数 resamples（）意指：针对模型性能的推理评估（Inferential Assessments About Model Performance）
```

结果：

```
R Console
> summary(RESULT)

Call:
summary.resamples(object = RESULT)

Models: lda, knn, svm, rf
Number of resamples: 10

Accuracy
      Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
lda 0.9333  0.9500 1.0000 0.9800       1    1    0
knn 0.9333  0.9333 1.0000 0.9733       1    1    0
svm 0.8667  0.9333 1.0000 0.9600       1    1    0
rf  0.8000  0.9298 0.9333 0.9395       1    1    0

Kappa
    Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
lda  0.9  0.9250    1.0 0.9700       1    1    0
knn  0.9  0.9000    1.0 0.9600       1    1    0
svm  0.8  0.9000    1.0 0.9400       1    1    0
rf   0.7  0.8942    0.9 0.9092       1    1    0

> |
```

图 6.10 比较不同算法的结果

```
# 绘制图像
dotplot(RESULT)

#输出规则
print(fit.lda)
```

结果：

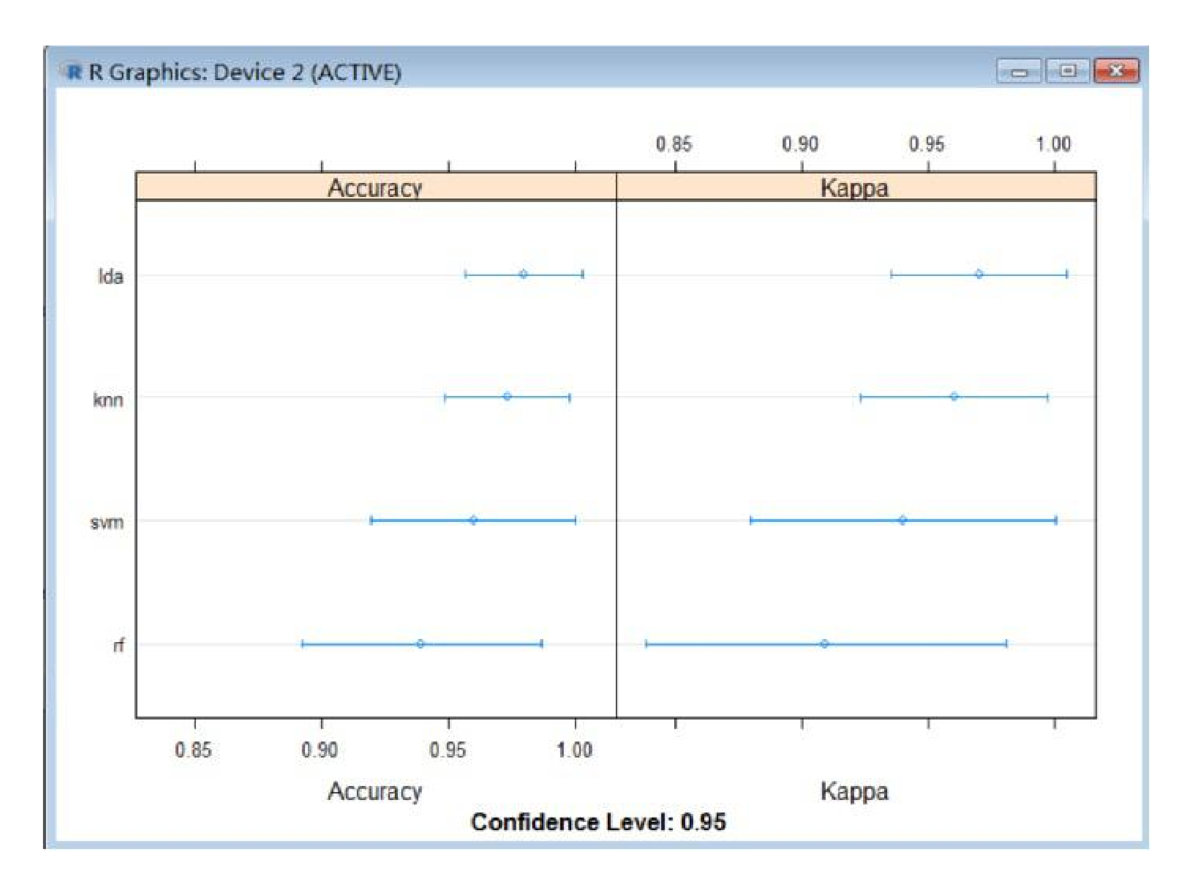


图 6.11. 不同算法的图像

```
> print(fit.lda)
Linear Discriminant Analysis

149 samples
  4 predictor
  3 classes: 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 134, 134, 134, 134, 135, 134, ...
Resampling results:

  Accuracy  Kappa
  0.98      0.97
```

图 6.12. 不同算法的规则

改良结果

```
# 定义预测
PREDICTION <- predict(fit.lda, nihao)

# 进行预测
confusionMatrix(PREDICTION, nihao$Iris.setosa)
## 函数 confusionMatrix()意指：输出用以估计所采样的混淆矩阵（Estimate a Resampled Confusion Matrix）的方式。
```

结果：

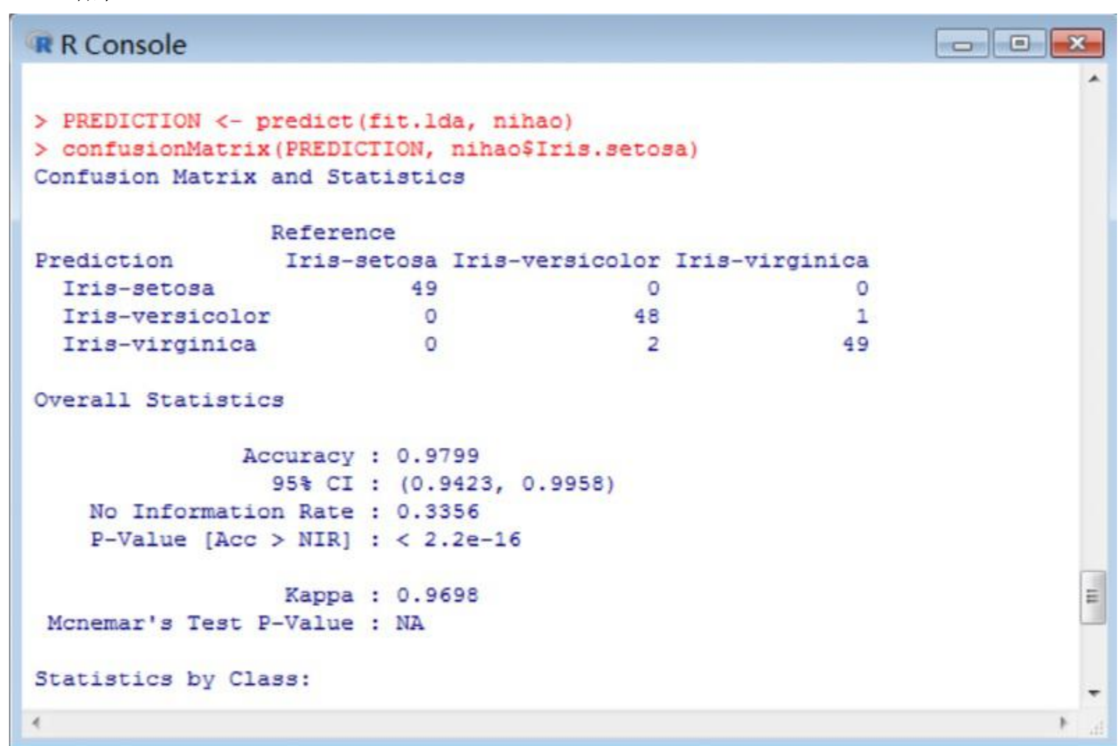


图 6.13. 进行预测

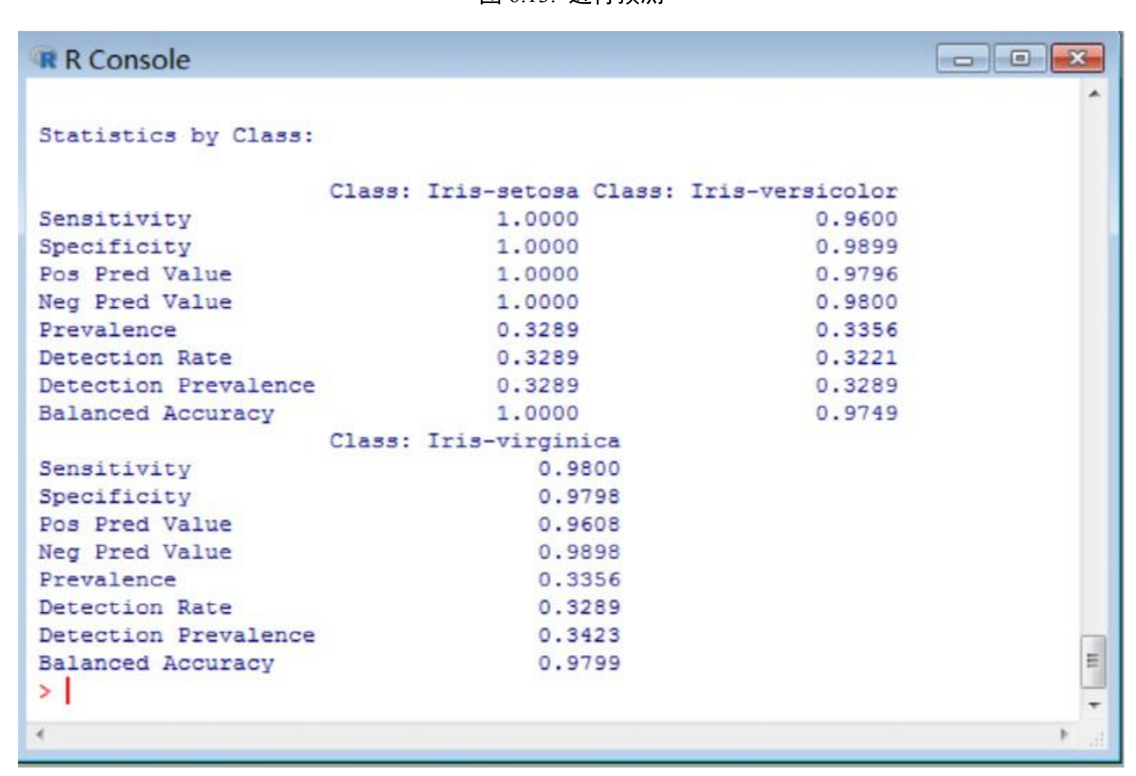


图 6.14. 预测结果与真实结果的对比

背景：通过专家知识，处理海量花朵数据。

研究问题：假如已知特征，如何可知类别。

解决方案：通过四种算法进行机器学习。

研究发现：此例 lda 精准度较高（如图 7-11）。
研究限制：应当包括更多算法试验（如贝叶斯、神经网络等）
结论：建立基于 lda 的花朵品种分类模型。

第二节 无监督学习

一、主成分分析

以下，为求便于快速理解，我们统一采取六个步骤，完成示例的练习。

1.研究目标设定（Setting the research goal）

2.检索数据（Retrieving data）

3.数据准备（data prepareation）

4.数据探索（data exploration）

5.数据建模（data modeling）

6.图表呈现（presentation）

利用【主成成分分析】分析农业生态经济数据

（一）主成分分析的含义:从原始变量（原始观测世界的众多指标）形成而来的新指标。它具有如下特性：

1.主成分保留原始变量绝大多数的信息；

2.主成分的个数少于原始变量的个数；

3.各个主成分之间互不相干；

4.每个主成分都是原始变量的线性组合。

（二）主成分分析的步骤：

1.计算相关系数矩阵；

2.计算相关系数矩阵的特征根以及所对应的特征向量；

3.选出最大的特征根，所对应的特征向量等于第一主成分的系数；

选出第二大的特征根，所对应的特征向量等于第一主成分的系数；

...以此类推。

4.计算累积贡献率，选择恰当的主成分的个数；

5.写出前 k 个主成分的表达式。

```
#1.2.备齐工具
#install.packages("labdsv") # 安装 labdsv 程序包，本例不安装，也可以完成主成分分析。
#library(labdsv)    # 调用 labdsv 程序包，本例不调用，也可从 R 语言里直接使用 princomp()进行分析

#1.3.设好路径
setwd("C:/Users/Hp/Desktop/MyData")
## 当前我的数据存放在 C:\Users\Hp\Desktop\MyData 里。
## 也可以写为 setwd("C:\\Users\\Hp\\Desktop\\MyData")

#2. 检索数据 Retrieving data
a=read.table("213.txt",header=TRUE) #读入数据集 213.txt
dim(a) # 看数据的维度，也就是规模，也就是多大；此例，数据类型是数据框，所以看有多少行、多少列。
```

```
head(a)  # 看前六行的数据。

#3. 数据准备 data prepareation
a=a[,-1]  #剔除第一列序号
dim(a)
head(a)
```

结果：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
> setwd("C:/Users/Hp/Desktop/MyData")
> a=read.table("213.txt",header=TRUE)
> dim(a)
[1] 21 10
> head(a)
  index      x1    x2     x3      x4     x5     x6     x7    x8     x9
1     1  63.912 0.352 16.101  192.11 295.34 26.724 18.492 2.231 26.262
2     2  41.503 1.684 24.301 1752.35 452.26 32.314 14.464 1.455 27.066
3     3  00.695 1.067 65.601 1181.54 270.12 18.266  0.162 7.474 12.489
4     4  43.739 1.336 33.205 1436.12 354.26 17.486 11.805 1.892 17.534
5     5  31.412 1.623 16.607 1405.09 586.59 40.683 14.401 0.303 22.932
6     6  68.337 2.032 76.204 1540.29 216.39  8.128  4.065 0.011  4.861
> a=a[,-1]
> dim(a)
[1] 21  9
> head(a)
       x1    x2     x3      x4     x5     x6     x7    x8     x9
1 363.912 0.352 16.101  192.11 295.34 26.724 18.492 2.231 26.262
2 141.503 1.684 24.301 1752.35 452.26 32.314 14.464 1.455 27.066
3 100.695 1.067 65.601 1181.54 270.12 18.266  0.162 7.474 12.489
4 143.739 1.336 33.205 1436.12 354.26 17.486 11.805 1.892 17.534
5 131.412 1.623 16.607 1405.09 586.59 40.683 14.401 0.303 22.932
6  68.337 2.032 76.204 1540.29 216.39  8.128  4.065 0.011  4.861
> 
```

图 6.15 主成分分析（1）

XXXXXX ?

```
#4. 数据探索 data exploration
summary(a)
```

结果：

R Console (64-bit)

文件 编辑 其他 程序包 窗口 帮助

```
6   68.337 2.032 76.204 1540.29 216.39   8.128   4.065 0.011   4.861
> summary(a)
       x1               x2               x3               x4        
 Min.   : 51.27   Min.   :0.352   Min.   :16.10   Min.   : 103.5  
 1st Qu.: 76.95   1st Qu.:0.731   1st Qu.:50.30   1st Qu.: 897.4  
 Median : 99.27   Median :0.841   Median :60.70   Median :1124.0  
 Mean   :113.08   Mean   :1.010   Mean   :55.02   Mean   :1067.0  
 3rd Qu.:131.41   3rd Qu.:1.245   3rd Qu.:66.50   3rd Qu.:1313.1  
 Max.   :363.91   Max.   :2.032   Max.   :76.20   Max.   :1752.3  
       x5              x6               x7               x8        
 Min.   :175.2   Min.   : 4.005   Min.   : 0.162   Min.   :0.0010  
 1st Qu.:196.4   1st Qu.: 9.521   1st Qu.: 4.066   1st Qu.:0.0120  
 Median :225.2   Median :17.486   Median : 5.643   Median :0.0550  
 Mean   :257.3   Mean   :16.643   Mean   : 6.721   Mean   :0.9053  
 3rd Qu.:270.1   3rd Qu.:19.409   3rd Qu.: 7.162   3rd Qu.:0.3030  
 Max.   :586.6   Max.   :40.683   Max.   :18.492   Max.   :7.4740  
       x9        
 Min.   : 3.201  
 1st Qu.: 5.402  
 Median : 8.413  
 Mean   :10.356  
 3rd Qu.:12.654  
 Max.   :27.066  
> 
```

图 6.16 主成分分析（2）

XXXXX?

```
#5. 数据建模 data modeling
a.pr=princomp(a,cor=TRUE)
##函数 princomp()进行主成成分分析
##函数 princomp()的第一个参数是数据集，第二个参数 cor 设为 FALSE 表示使用协方差矩阵，设为 TRUE 表示用相关系数矩阵
options(digits=4)  #结果显示 4 位有效数字
summary(a.pr,loadings=TRUE)
##loadings=TURE 表示显示载荷系数，即：主成分对应原始变量线性组合的系数。
```

结果：

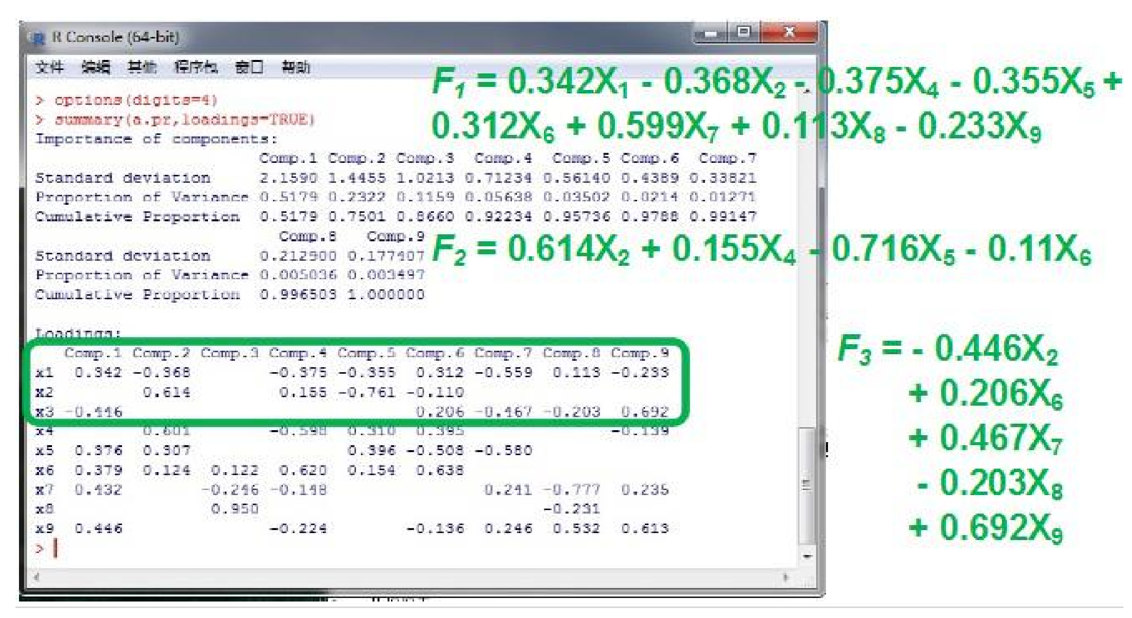


图 6.17 主成分分析（3）

所以，我们用 F1 、F2、F3 来取代 X1 、X2 、X3 、X4 、X5 、X6 、X7 、X8 、X9

```
#6. 图表呈现 presentation
screeplot(a.pr,type="line",main="碎石图")
biplot(a.pr)
```

结果：

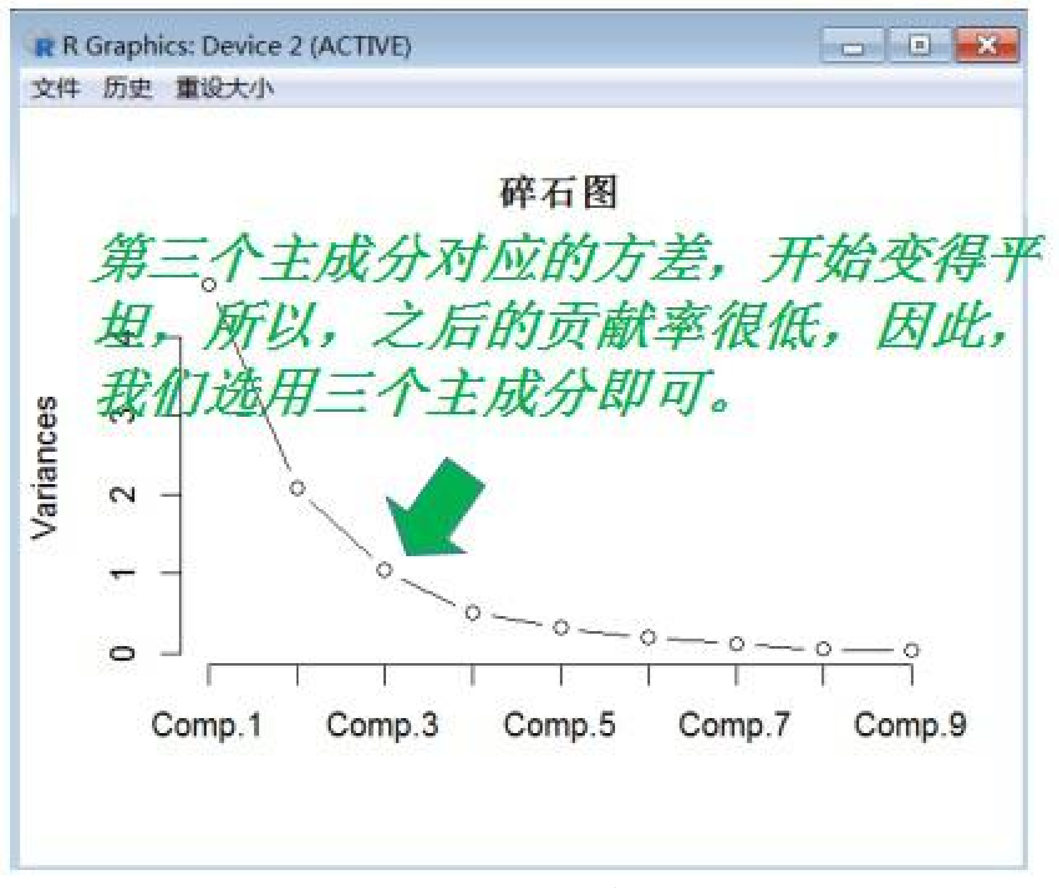


图 6.18 主成分分析（4）

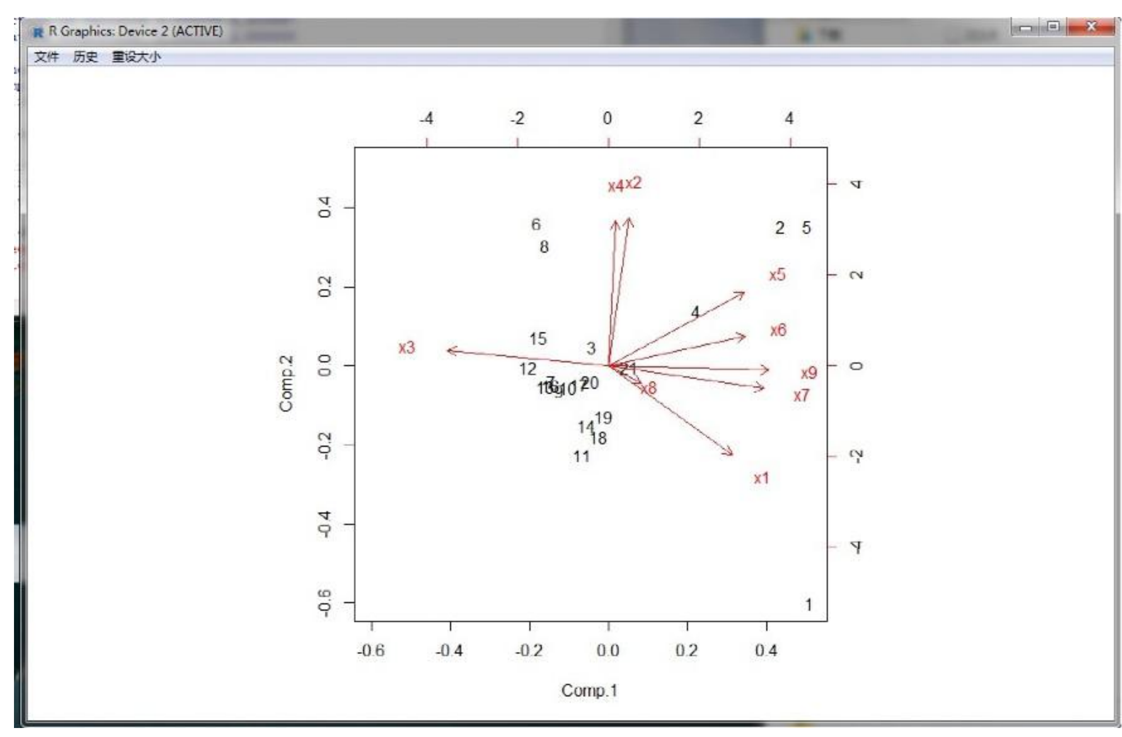


图 6.19 主成分分析（5）

现在我们再次尝试一组数据集，利用【主成成分分析】分析各地农村居民家庭日均主食消费量。所采用的方法是 PCA 方法。

```
#1.备齐工具、设好路径
#install.packages("labdsv") # 如果之前没有安装，现在就需要安装。
library(labdsv) # 调用 labdsv 程序库之后，可以使用函数 pca()进行主成分分析。
setwd("C:/Users/Hp/Desktop/MyData")

#2. 检索数据 Retrieving data
b=read.table("214.txt",header=T)
dim(b)
head(b)

#3. 数据准备 data prepareation
b=b[,-1]
dim(b)
head(b)

#4. 数据探索 data exploration
summary(b)

#5. 数据建模 data modeling
b.pca=pca(b,dim=4,cor=TRUE)
##函数 pca()的第一个参数 mat 必需是数据框或者矩阵。
##把函数 pca(mat,dim,cor)的参数 cor 设为 TRUE 表示利用【相关系数矩阵】进行计算（主成

分分析）
##把函数 pca(mat,dim,cor)的参数 dim 设为 4 表示变量综合之后的维度是 4(所以主成分为
```

```
3)

summary(b.pca)
loadings.pca(b.pca) #使用函数 loadings.pac()提取载荷系数。
```

结果：

```
R Console (64-bit)
文件 编辑 其他 程序包 窗口 帮助
 Mean   : 9.95
 3rd Qu.:13.64
 Max.   :24.15
> b.pca=pca(b,dim=4,cor=TRUE)
> summary(b.pca)
Importance of components:
                            [,1]      [,2]      [,3]       [,4]
Standard deviation     1.7108133 1.4900949 1.1594286 0.89521303
Proportion of Variance 0.3252091 0.2467092 0.1493638 0.08904515
Cumulative Proportion  0.3252091 0.5719183 0.7212822 0.81032730
> loadings.pca(b.pca)

Loadings:
   PC1    PC2    PC3    PC4
X1        -0.550        -0.242
X2        -0.320 -0.616  0.457
X3         0.186 -0.697 -0.470
X4  0.456        -0.168  0.505
X5  0.509 -0.142        -0.263
X6 -0.329  0.408 -0.270
X7  0.501  0.119 -0.112 -0.367
X8  0.388  0.333  0.130  0.113
X9  0.141  0.493         0.201
>
```

图 6.20 主成分分析（6）

第一主成分，对 X4～X8 的荷载系数较大。

第二主成分，对 X1、X6、X9 的荷载系数较大。

第三主成分，对 X2、X3 的荷载系数较大。

第四主成分，反映的是全部变量的综合情况。

```
#6. 图表呈现 presentation
op=par(mfrow=c(1,2))  #分割图形区域为 1 行 2 列
varplot.pca(b.pca)
layout(1) # 方差贡献图 Variance
```

结果：

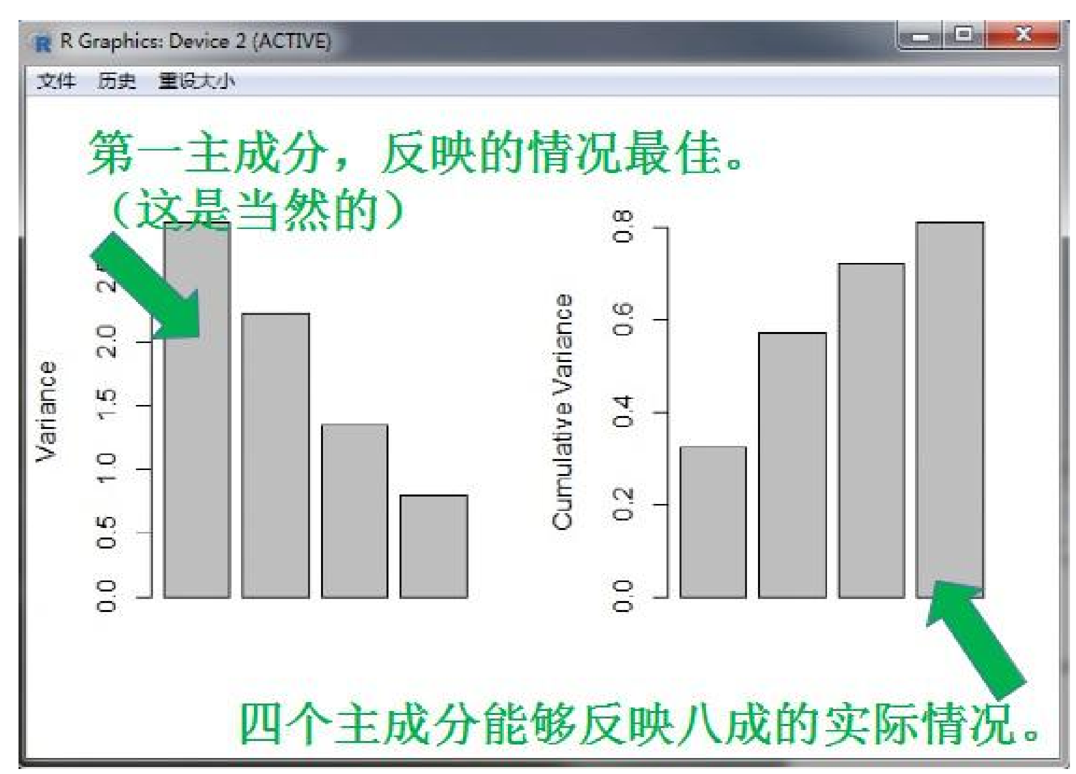


图 6.21 主成分分析（7）

二、因子分析

因子分析的含义:所观察的变量的值，具有聚合趋势，彼此之间存在高度相关性，把高度相关性来源于一个共同的制约因素就是共同因子。我们这节使用【因子分析】分析商业银行财务指标，得到代表银行经营绩效的指标体系。

因子分析的步骤

（一）设定作为因子分析的原始变量。

（二）构造因子载荷矩阵。

（三）旋转因子矩阵计算因子方差、方差贡献、累计方差贡献等，

（四）形成公共因子。

在进行分析之前我们构造一组【函数】用来进行之后的分析。

```
FA=function(x,m){
  p=nrow(x);x.diag=diag(x);sum.rank=sum(x.diag)
  rowname=paste("变量 X",1:p,sep="")
  colname=paste("因子 Factor",1:m,sep="")
  A=matrix(0,nrow=p,ncol=m,dimnames=list(rowname,colname))
  eig=eigen(x)
  for(i in 1:m)
    A[,i]=sqrt(eig$values[i])*eig$vectors[,i]
  var.A=diag(A%*%t(A))
  rowname1=c("两主成分的特征值 SS loadings","方差比例 Proportion Var","累计方差比例 Cumulative Var")
  result=matrix(0,nrow=3,ncol=m,dimnames=list(rowname1,colname))
  for(i in 1:m){
```

```
    result[1,i]=sum(A[,i]^2)
    result[2,i]=result[1,i]/sum.rank
    result[3,i]=sum(result[1,1:i])/sum.rank
  }
  method=c("主成分方法 Principal Component Method")
list(method=method,loadings=A,var=cbind(common=var.A,specific=x.diag-var.A),result=result)
}
```

上述代码的注解，如下：

构造一个函数，第一位参数 x（往后我们设置为矩阵），第二位参数 m（往后我们设为因子数）
构造一个对角矩阵 diag 及其元素，这里定义了函数 FA 的第一个参数。
##设置行名，以 x 起首。
##设置列名，以 Factor 起首。
##构造因子载荷矩阵 A，初值设为 0
##函数 eigen（）是基于特征向量的计算，处理统计学里面的共线性问题。 #eig 包含两个元素,values 为特征根，vectors 为特征向量。
##这里的 m 在代码里是 for 循环次数，但在我们的函数 FA 里也就是指因子个数。
##填充矩阵 A 的值。
##求取公共因子的方差，对角函数 diag()里矩阵 A 和转置矩阵 A 的相乘后的方差。

##构造输出结果的矩阵，初值设为 0
##计算各因子的方差
##计算方差贡献率
##累计方差贡献率

##输出计算结果
##loadings 载荷系数；var 方差；diag 对角矩阵

```
1.设定路径
setwd("C:/Users/Hp/Desktop/MyData")

2. 检索数据 Retrieving data
a=read.table("215.txt",header=T)
dim(a) # 看数据的维度，也就是规模，也就是多大。
head(a) # 看前六行的数据。

#3. 数据准备 data prepareation
a=a[,-1]  #剔除第一列序号
dim(a)
head(a)

#4. 数据探索 data exploration
summary(a)

#5. 数据建模 data modeling
```

```
b=cor(a)  #计算相关系数矩阵
options(digits=3)  #结果显示 3 位有效数字
```

结果：

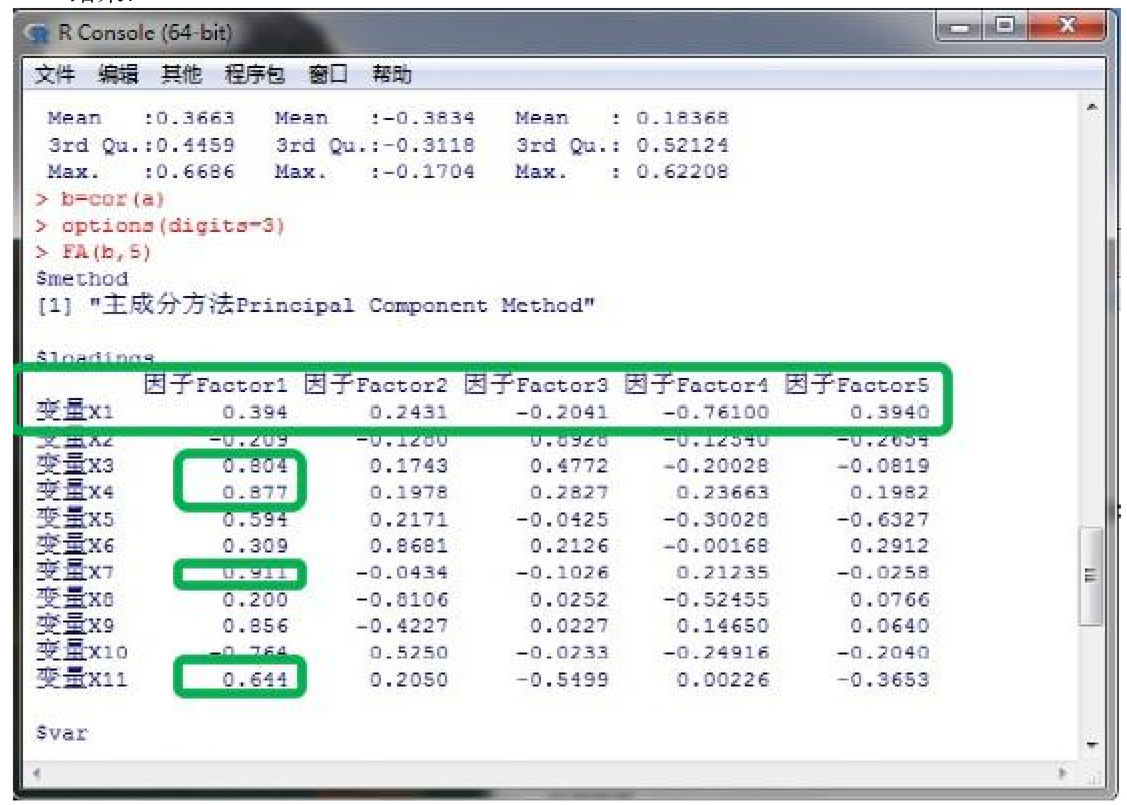


图 6.22 因子分析（1）

我们根据图 6.22 得知两个重点：

（一）原始变量 X1 和五个因子变量的线性关系

X1 = 0.394F1 + 0.2431F2 - 0.2041F3 -0.761F4 + 0.394F5

（二）因子变量 Factor1 与 X3、X4、X7 和 X9 的载荷系数均大于 0.8 呈现高度正相关。

因此，我们进行最后一个步骤：图表呈现。这里，我们选择五个因子，用到之前设计的因子分析的函数。

```
FA(b,5)
```

结果：

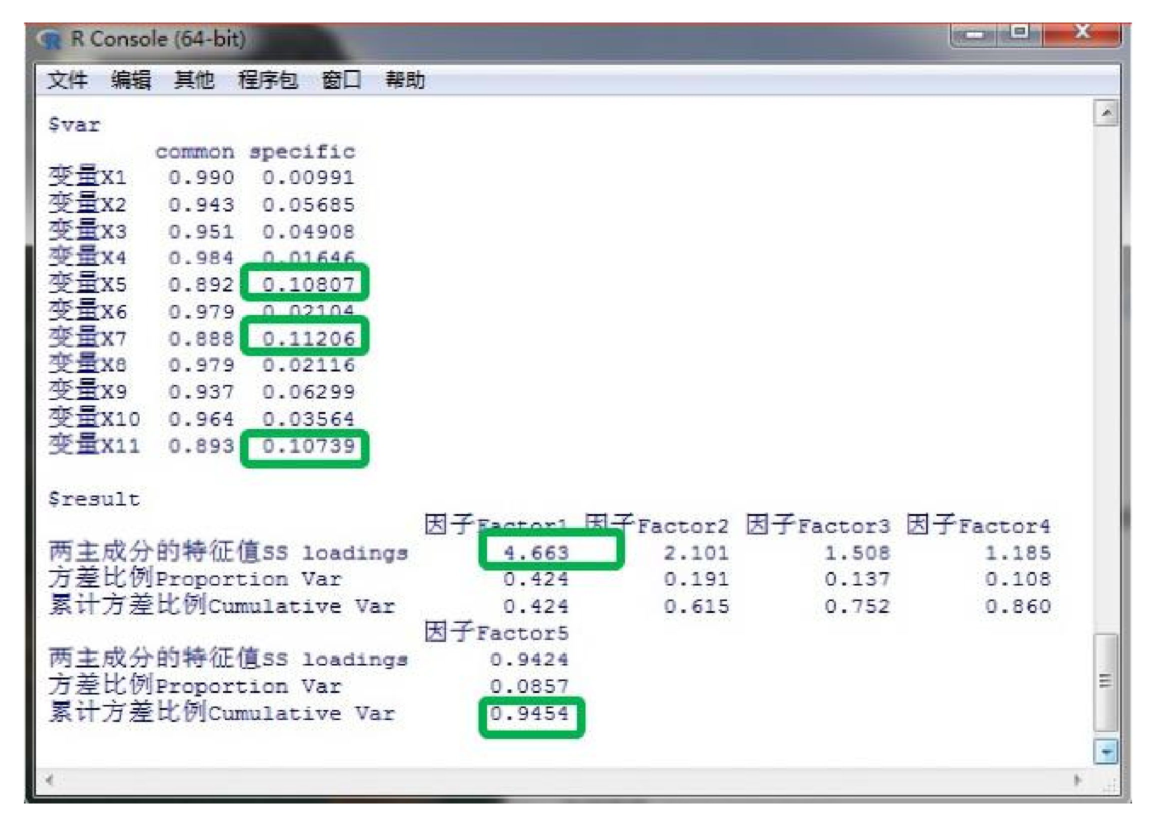


图 6.23 因子分析（2）

解读图 6.23 的内容，有三个重点：

（一）变量 5、7、11 有明显偏向某些因子变量的情况。（因子 5、因子 1、因子 1）

（二）第一个因子变量，占比最高。

（三）所选取的 5 个因子变量，反映了大部分信息（累计贡献率 94.54%）

三、因子分析与主成分分析的综合用法

我们使用【因子分析】结合【主成成分分析】从消费者问卷结果中，简化变量，得到解释。以下示例，采用李斯特量表（1 表示非常不同意，7 表示非常同意）记录消费者的数据，有 6 列（变量）。我们把 7 等分的量表数据，转化为 2 个变量。

```
#1.备齐工具
#install.packages("labdsv")
library(labdsv)
setwd("C:/Users/Hp/Desktop/MyData")

#2. 检索数据
a = read.table("216.txt",header=TRUE)
dim(a)
head(a)

#3. 数据准备
a=a[,-1]
dim(a)
head(a)
```

```
#4. 数据探索
summary(a)
# 目前为止，是消费者记录中，各个变量的情况。

#5. 数据建模
factanal(a,factors=2)
b=pca(a,dim=2,cor=TRUE)
## 因子分析：两主成分的特征值 SS loadings、方差比例 Proportion Var、累计方差比例 Cumulative Var
## 主成分分析：标准偏差 Standard deviation、 方差比例 Proportion Var、累计方差比例 Cumulative Var
```

结果：

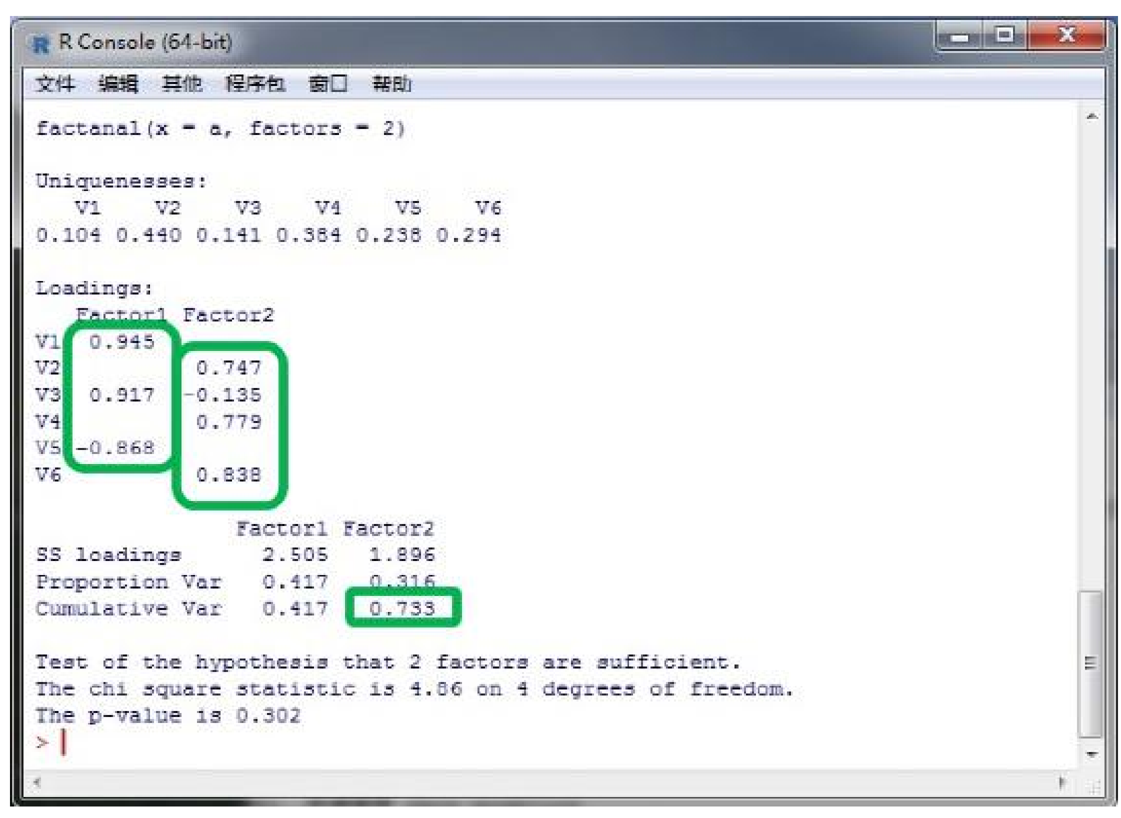


图 6.24 因子分析与主成分分析的综合应用（1）

得到两个因子

F1 = 0.945V1 + 0.917V3 - 0.868V5

F2 = 0.747V2 - 0.135V3 + 0.779V4 +0.838V6

其累计贡献量 0.735 所以，我们进行最后一个步骤：图表呈现。

```
summary(b)
plot(b)
# 现在是经过因子分析和主成分分析后，代表消费者的两个“人为”制造出来的变量。
# 根据这两个变量，组成 x 轴与 y 轴，把样本放在二维空间里，感受到的三组消费者群体。
```

R Console (64-bit)

文件 编辑 其他 程序包 窗口 帮助

```
V1  0.945
V2          0.747
V3  0.917  -0.135
V4          0.779
V5 -0.868
V6          0.838

               Factor1 Factor2
SS loadings      2.505   1.896
Proportion Var   0.417   0.316
Cumulative Var   0.417   0.733

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 4.86 on 4 degrees of freedom.
The p-value is 0.302
> b=pca(a,dim=2,cor=TRUE)
> summary(b)
Importance of components:
                            [,1]      [,2]
Standard deviation     1.6506047 1.4834355
Proportion of Variance 0.4540826 0.3667635
Cumulative Proportion  0.4540826 0.8208461
> plot(b)
>
```

累积方差比超过0.8比较能够代表概括了整体情况。

图 6.25 因子分析与主成分分析的综合应用（2）

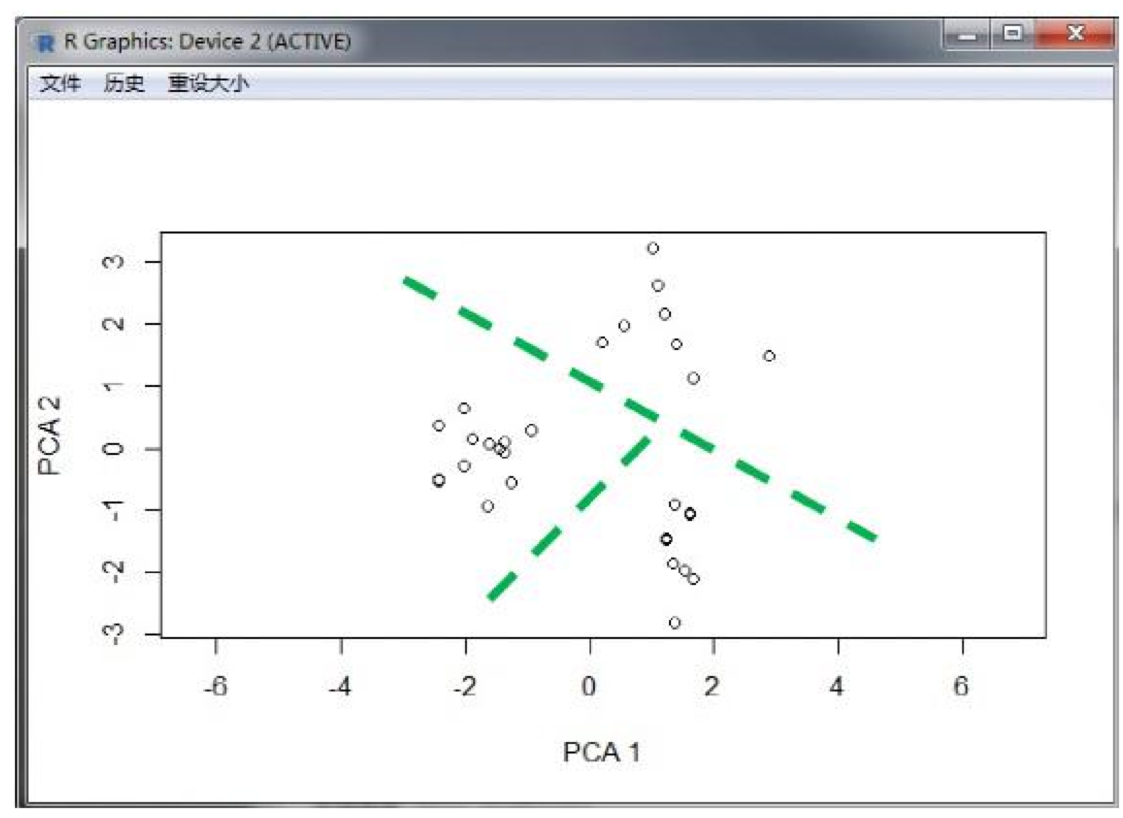


图 6.26 因子分析与主成分分析的综合应用（3）

四、关联分析

关联分析的含义是：两组变量之间的相关（而不是两个变量），根据变量间的相关性，把两组变量的关系集中到少数综合变量上。关联分析的步骤，包括：

（一）原始数据的标准化处理

（二）计算典型相关系数

（三）根据载荷系数写出各组变量的表达式

（四）基于领域知识的理论解释。

此处，我们使用【关联分析】分析财务数据，，求取两组变化之间的关系。

```
#1.设好路径
setwd("C:/Users/Hp/Desktop/MyData")

#2. 检索数据
a=read.table("215.txt",header=T)
dim(a)
head(a)

#3. 数据准备
a=a[,-1]  #去除第一列（通常为年份或序号）
summary(a)

#4. 数据探索
a=scale(a)  #对数据作正态离差标准化
summary(a)

#5. 数据建模
ca=cancor(a[,1:4],a[,5:8],xcenter=TRUE,ycenter=TRUE)
##函数 cancor()的前两个参数是两组变量的数据矩阵。
##函数 cancor()的后两个参数是数据中心化，默认值为 TRUE。
options(digits=4)

#6. 图表呈现
ca
```

结果：

```
R Console (64-bit)
文件  编辑  其他  程序包  窗口  帮助

$cor
[1] 0.9984 0.9512 0.4436 0.3557

$xcoef
          [,1]    [,2]     [,3]     [,4]
X1  0.02705 -0.1800 -0.01501 -0.24812
X2 -0.05352 -0.2462 -0.67839  0.08167
X3 -0.13291  1.6587  1.86941 -1.41084
X4 -0.11931 -1.5133 -1.37607  1.38437

$ycoef
          [,1]    [,2]    [,3]    [,4]
Y1 -0.08637 -0.1173 -0.7481  2.2384
Y2  0.06487  1.2353  0.8194  4.7401
Y3 -0.05893  1.1485 -9.3603 -6.8260
Y4 -0.20940 -2.2606 -0.7167 -0.1437

$xcenter
        X1         X2         X3         X4
 1.308e-16  5.658e-17  5.871e-18 -6.192e-17

$ycenter
        Y1         Y2         Y3         Y4
 2.178e-16  7.686e-17 -9.714e-17 -1.183e-16
```

图 6.27 关联分析结果

从图 6.27 我们可以得到两组关联关系

第一种关联关系是：

第一组变量 U1 = 0.027X1 - 0.054X2 - 0.133X3 - 0.119X4

第二组变量 V1 = -0.086Y1 + 0.065Y2 - 0.059Y3 - 0.209Y4

第二种关联关系是：

第一组变量 U2 = -0.18X1 - 0.246X2 + 1.659X3 - 1.513X4

第二组变量 V2 = -0.117Y1 + 1.235Y2 + 1.149Y3 - 2.261Y4

上述表达式，其变量之前的数值，即为：载荷系数。因子轴旋转之后，对应到各个变量的系数，前四种因子旋转轴下的各组参数，我们首选第一组，并且对照第二组，来看差异性。

在第一种关联关系上，第一组变量和第二组变量的相关关系是 0.9984 高度相关，在第二种关联关系上，其相关关系是 0.9512 高度相关，但是其后的 0.4436 和 0.3557 使得我们决定只需要看第一种和第二种的关联关系。

通过比较，又会发现，U1 和 U2 以及 V1 和 V2 相比是差距很大的，即使 U1 和 V1 以及 U2 和 V2 在各自的关联关系里能够反应整体数据集的情况。据此，我们可以知道，关联分析的这种方式，能够探索出一种解释现象的模型，但是这种模型选择的参数差异很大。

我们如果只是在小数据的范围内，那么可以很快解决问题，但是不能求得一个相对稳定的模型参数，所以，如果是在大数据的范畴下（第七章），那么如果能够解决快速计算以及实时数据更新的工程难题，那么就有可能在不用顾及模型参数是否稳定的情况下，直接解决问题（第五章所说的暴力破解的建模方式，能否长期有效的讨论）。

第三节 有监督学习

一、贝叶斯学习

（一）基本原理

朴素贝叶斯是依据概率原则进行分类的机器学习算法，它通过对于过去已经发生时间的数据来推断未来事件发生的概率。

在 18 世纪，托马斯贝叶斯（Thomas Bayes）发明了用于描述事件的概率，以及如何根据附加信息修改概率的基本数学原理；简言之，一个事件发生的概率是一个介于 0 到 100%之间的数。

根据贝叶斯方法建立的分类器，它利用训练集以及特征取值，来计算每个类别被观察到的概率；当分类器处理无标签的数据时，它会根据观测到的概率来预测新的特征最有可能属于哪个类别。

贝叶斯概率理论的思想：一个事件（event）的似然估计应该建立在已知已有的证据的基础上；事件就是可能的结果，试验（trial）就是事件发生一次的机会。

（二）特性：结合先验概率与后验概率

概率：事件发生的试验的次数，除以试验的总次数。

联合概率：表示两个事件共同发生的概率；一个事件发生的概率，与另一个事件发生的概率，如果是相关事件，则可建立预测模型，如果是独立事件，则不能根据已有的事件的数据来预测另一个事件发生的概率。

条件概率：在事件 B 已经发生的条件下，事件 A 发生的概率；事件 A 发生的概率依赖于事件 B 的发生（条件）。

贝叶斯定理：作为描述相关事件之间的关系的一种方式；它不仅考虑到事件 A 发生的概率，也考虑事件 B 发生的概率以及对于事件 A 发生的影响，并且又考虑到 A 和 B 同时发生的频率，作为试验对于事件 A 发生的概率的修正。

（三） 贝叶斯定理：

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

图 6. 28 贝叶斯定理

P(A)：先验概率，在没有任何附加证据的条件下，事件 A 发生的概率。

P(B | A)：似然概率，根据发生 A 并且发生 B 的频率，所建立的概率。

P(B)：边际似然概率，作为引发事件 A 发生的事件 B 所可能发生的概率。

P(A | B)：后验概率，用来衡量发生的可能性，若大于 50%则判断发生。

（四）朴素贝叶斯的假定

朴素贝叶斯（Naïve Bayes, NB）：假设数据集的所有特征都具有独立性并且重要性相同。

类条件独立（class-conditional independence）:事件在相同类取值的条件下，事件就是相互独立；朴素贝叶斯的这个条件独立性的假设，允许人们应用独立事件的概率原则来简化计算公式。

（五）朴素贝叶斯的定义

$$P(C_L|F_1,...,F_n) = \frac{1}{Z}p(C_L)\prod_{i=1}^{n}p(F_i|C_L)$$

图 6.29 朴素贝叶斯

P(Fi|CL)： 每条证据 F（第 1 条到第 n 条）在类 C 水平为 L 的概率

CL：类 C 水平为 L 的概率。

Π：乘积

P(CL)：类 C 水平 L 的先验概率。

1/Z：尺度因子，把结果转换为一个概率值。

P(CL| F1…Fn)：在给定特征的条件下，类 C 水平为 L 的概率。

F1…Fn：给定特征的条件。

（六）朴素贝叶斯的细节：拉普拉斯估计

根据朴素贝叶斯算法，概率是相乘得到一个介于 0 和 100%之间值，所以如果有某个证据 F 的发生概率为 0 的情况，将导致后验概率为 0 的值，从而抵消或者否决其他所有证据。为此，需要考虑如何使得每一类中每个特征发生的概率是非零。

拉普拉斯估计（Laplace estimator）：是用来反映一个事先假定的先验概率，这个概率涉及特征 F 和类 CL 之间的联系情况。

拉普拉斯估计所增加的数，可以是任何一个值，在实际应用上，在数据集较大也就是证据较多的前提下，通常设定为 1 然后在频率表中的每个计数加上这个数，保证每一类特征的组合至少在数据集中出现一次。

（七）朴素贝叶斯的细节：数值特征离散化

数值特征离散化：使用朴素贝叶斯算法时，每个特征必须是分类变量，才能创建类和特征值的组合所构成的矩阵，然而数值型特征没有分类变量的类别值，所以需要把特征的颗粒度降低，划分到几个类别里面。

离散化的类：把数值划分到不同的分段（bin）里，有两种方式：

1.数据探索的方式：人们探索用于自然分类的数据，或者数据分布中的分割点。

2.分位数的方式：例如利用四分位数，把数据分到四个分段里。

研究者需要注意平衡：分段太少，容易掩盖重要趋势，分段太多，容易导致朴素贝叶斯的频率表的计数值很小。

（八）研究问题与数据采集

垃圾短信的自动过滤问题：某些广告商利用短信服务（SMS）文本信息，面向终端用户，发送不需要的广告（即：垃圾短信）。早期人们需要为收到手机短信付费，这样容易引起电信服务商与消费者之间的付费纠纷，现今则是太多推送信息，影响用户体验。

另一方面，垃圾短信的文本信息相对垃圾邮件更少，而且短信信息通常采用缩写或者术语，这样，垃圾短信和正常短信的区别就更不容易。

以下使用一组已经整理好的数据集，然而我们在这里也稍作一些修改，以便通过朴素贝叶斯算法，建立一个分类器，然后我们希望能够根据这个数据集里的短信信息，把它所有单词提供的证据，用来计算垃圾短信和正常短信的概率，最后做出自动过滤机制。

（九）数据检查和处理

我们需要把文本信息，转化为计算机能够理解的形式，那些词和句子在这里要被转化为词袋（bag-of-words）的表示方法，我们忽略单词出现的顺序，而只关注单词是否出现。

```
getwd() # 查看路径
setwd("D://mydata")  # 设定路径，这里之前我们把数据集放在 mydata 了。

sms_raw <- read.csv("SMSSpamCollection.csv", stringsAsFactors = FALSE) # 导入数据
str(sms_raw) # 看看 sms 数据的结构
sms_raw$type <- factor(sms_raw$type)# 转换为因子 factor 变量
str(sms_raw$type) # 再次检查一遍变量类型

install.packages("tm") # 安装 tm 文本挖掘程序包
library(tm) #调用 tm 这个程序包
sms_corpus <- Corpus(VectorSource(sms_raw$text))

# 建立一个训练数据有短信的语料库，即：文本文件的集合。
## 函数 Corpus()可以读入许多不同来源的文档，这里是使用它的 VectorSource 参数，读取向量 sms_raw$text 的信息，然后把结果存储到 sms_corpus 对象里去。

print(sms_corpus) # 查看有 5574 条短信。
inspect(sms_corpus[1:3]) # 查看第 1、2、3 条短信的具体内容
```

结果：



图 6.30 数据检查和处理

（十）数据清洗

我们用 tm_map()函数清洗数据，通过映射的方式来整理语料库。

```
corpus_clean <- tm_map(sms_corpus, tolower)
## 全部字母变成小写字母。
corpus_clean <- tm_map(corpus_clean, removeNumbers)
## 清除所有数字。
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
## 清除停用词，如 to、and、but、or 等。
corpus_clean <- tm_map(corpus_clean, removePunctuation)
## 清除标点符号。
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
## 上述这些内容清除后，会留下空格，所以清除额外空格，只留下词与词之间的空格。
```

接着上述代码，下面代码，主要作用是建立一个清洗过的词汇的数据库：

```
# 查看新的语料库 sms_corpus 也就是数据清洗后的数据集。
head(sms_corpus)

# 比较清洗前后，通过函数 inspect()以及访问向量的方式，查看前三条短信内容。
inspect(sms_corpus[1:3])
inspect(corpus_clean[1:3])

# 创建一个稀疏矩阵（sparse matrix），把语料库标记化，之后就可进行分析。一个记号
（token）就是一个文本字符串的元素，这里就是单词。
```

```
sms_dtm <- DocumentTermMatrix(corpus_clean)
sms_dtm
```

结果：

```
R Console
> corpus_clean <- tm_map(corpus_clean, removePunctuation)
> corpus_clean <- tm_map(corpus_clean, stripWhitespace)
> head(sms_corpus)
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 6
> inspect(sms_corpus[1:3])
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3

[1] Go until jurong point, crazy.. Available only in bugis n great world la $
[2] Ok lar... Joking wif u oni...                                            $
[3] Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text$
> inspect(corpus_clean[1:3])
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3

[1] go jurong point crazy available bugis n great world la e buffet cine got$
[2] ok lar joking wif u oni                                                  $
[3] free entry wkly comp win fa cup final tkts st may text fa receive entry $
> |
```

图 6.31 清洗数据

接着，我们需要创建【训练集】与【测试集】以便接下来的【有监督学习】的建模和检验。

接着上述代码，下面代码，主要作用是建立一个清洗过的词汇的数据库：

```
sms_raw_train <- sms_raw[1:4169, ] #分解原始的数据框
sms_raw_test  <- sms_raw[4170:5574, ]
sms_dtm_train <- sms_dtm[1:4169, ] #输出【文档-单词】矩阵
sms_dtm_test  <- sms_dtm[4170:5574, ]
sms_corpus_train <- corpus_clean[1:4169] #得到语料库
sms_corpus_test  <- corpus_clean[4170:5574]
```

比较垃圾短信在训练集与测试集的比例，确认为一组完整的短信信息数据：

```
prop.table(table(sms_raw_train$type))
prop.table(table(sms_raw_test$type))
```

结果：

```
R Console
[3] free entry wkly comp win fa cup final tkts st may text fa receive entry $
> sms_dtm <- DocumentTermMatrix(corpus_clean)
> sms_dtm
<<DocumentTermMatrix (documents: 5574, terms: 7997)>>
Non-/sparse entries: 43529/44531749
Sparsity           : 100%
Maximal term length: 40
Weighting          : term frequency (tf)
> sms_raw_train <- sms_raw[1:4169, ]
> sms_raw_test  <- sms_raw[4170:5574, ]
> sms_dtm_train <- sms_dtm[1:4169, ]
> sms_dtm_test  <- sms_dtm[4170:5574, ]
> sms_corpus_train <- corpus_clean[1:4169]
> sms_corpus_test  <- corpus_clean[4170:5574]
> prop.table(table(sms_raw_train$type))

      ham      spam
0.8647158 0.1352842
> prop.table(table(sms_raw_test$type))

      ham      spam
0.8697509 0.1302491
> |
```

图 6.32 比较训练集和测试集的比例

如图 6.32 所示，目前，训练集与测试集，都包括 13%左右的垃圾短信，表示垃圾短信被平均分配在两个数据集里面。

我们接下来要进行实际的分组，并且通过【词云】观察那些被分类了的词组。

比较垃圾短信在训练集与测试集的比例，确认为一组完整的短信信息数据：

```
# 载入词云可视化的程序包
install.packages(“wordcloud”); library(wordcloud)
wordcloud(sms_corpus_train, min.freq = 30, random.order = FALSE)
## 这里设定 FALSE，所以词云会以非随机方式排列，词频越高，越靠中心。

# 把训练集数据进行垃圾短信和正常短信的群组归类
spam <- subset(sms_raw_train, type == "spam")
ham  <- subset(sms_raw_train, type == "ham")

# 用词云可视化训练集里的垃圾短信和正常短信
wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
## 这里设定：出现最常见的 40 个单词，以及最大字体和最小字体。
```

结果：



图 6.33 词云

接下来，我们要为频繁出现的单词创建指标特征，以减少特征数量。

```
findFreqTerms(sms_dtm_train, 5)
# 利用 tm 程序包的 findFreqTerms()函数，以文档-单词矩阵的方式查找频繁出现的单词，并且返回一个字符向量（次数不少于指定次数的单词），这里设定为出现 5 次的单词。

sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
str(sms_freq_words)

# 训练集与测试集包括 1200 个特征，这 1200 个特征是至少出现在 5 条短信的单词
sms_dtm_freq_train <- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

结果：

```
R Console
[1177] "pple"           "arrested"       "croydon"
[1180] "fantasies"      "cover"          "write"
[1183] "fact"           "cafe"           "alone"
[1186] "loved"          "hows"           "arrive"
[1189] "screaming"      "auction"        "difficult"
[1192] "gas"            "excellent"      "john"
[1195] "instead"        "buzz"           "lei"
[1198] "waking"         "sight"          "father"
[1201] "hook"           "slowly"         "joys"
[1204] "holding"        "exciting"       "btnationalrate"
[1207] "wednesday"      "thnk"           "excuse"
[1210] "season"         "thinkin"        "mon"
[1213] "sitting"        "pix"            "colleagues"
[1216] "mood"           "sofa"           "empty"
[1219] "checked"        "sky"            "housemaid"
[1222] "murdered"       "murderer"       "police"
[1225] "budget"         "happens"        "thurs"
> sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
> str(sms_freq_words)
 chr [1:1227] "available" "bugis" "cine" "crazy" "got" "great" "point" ...
> sms_dtm_freq_train <- sms_dtm_train[ , sms_freq_words]
> sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
> |
```

图 6.34 减少特征数量后的数据集

接着，我们要设定一个函数，把计数转换为因子，把单词是否出现，表示为“Yes”或者“No”的因子水平。

```
convert_counts <- function(x) {
  x <- ifelse(x > 0, 1, 0)
  x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
}
# 函数 apply()包括 lapply()和 sapply()两个函数，它可以对 R 数据结构的每个元素进行操作，它有时比循环（for 和 while 语句）更高效。
# 函数 apply()的参数 MARGIN 用来指定矩阵的行列（行=1；列=2）

sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)

class(sms_train)
str(sms_train)
class(sms_test)
str(sms_test)
```

如此一来，我们就有两个矩阵，每个矩阵带有因子类型的列，用 Yes 和 No 代表每一列的单词是否出现在构成行的信息中。

这样，就把原始的短信信息，转换成为一个可以用统计模型代表的形式，可以采用朴素贝叶斯算法估计一条短信是否为垃圾短信。

（十一） 训练模型

这里我们使用 naiveBayes()函数，关于这个函数，需要重点介绍两项功能。

一是创建分类器，其指令类似 m <- naiveBayes(train, class, laplace=0) 是指：
1.train 数据框或者包含训练集的矩阵。
2.class 包含训练集的每一行的分类的一个因子向量。
3.laplace 控制拉普拉斯估计的一个数值（默认为零）。
二是用于预测，其指令类似 p <- predict(m, test, type = "class") 是指：
1.m 由函数 naiveBayes()所训练的模型。
2.test 数据框或者具有测试集的矩阵，具有与建立分类器的训练集的相同特征。
3.type 为 class 或者 raw 值，标识着预测的类别值或者预测概率。

　　该函数会返回一个向量，根据参数 type 值，向量含有预测的类别值，或者原始预测的概率值。

　　承接前面第（十）部分的代码，我们接下来进行训练模型的步骤，其代码为：

```
# 安装 1071 程序包（也有别的程序包）
install.packages("e1071")
library(e1071)

# 基于 sms_train 矩阵建立模型，使得 sms_classifier 包括一个可以用于预测的 naiveBayes 分类器对象。
sms_classifier <- naiveBayes(sms_train, sms_raw_train$type)
sms_classifier

## 现在 sms_classifier 对象，包括了一个可以用于预测的 naiveBayes 分类器的对象。它所产生的预测值，可以用来和真实值进行比较，从而得知是否分类器能够发挥机器学习的作用，形成自动过滤机制。
```

结果：

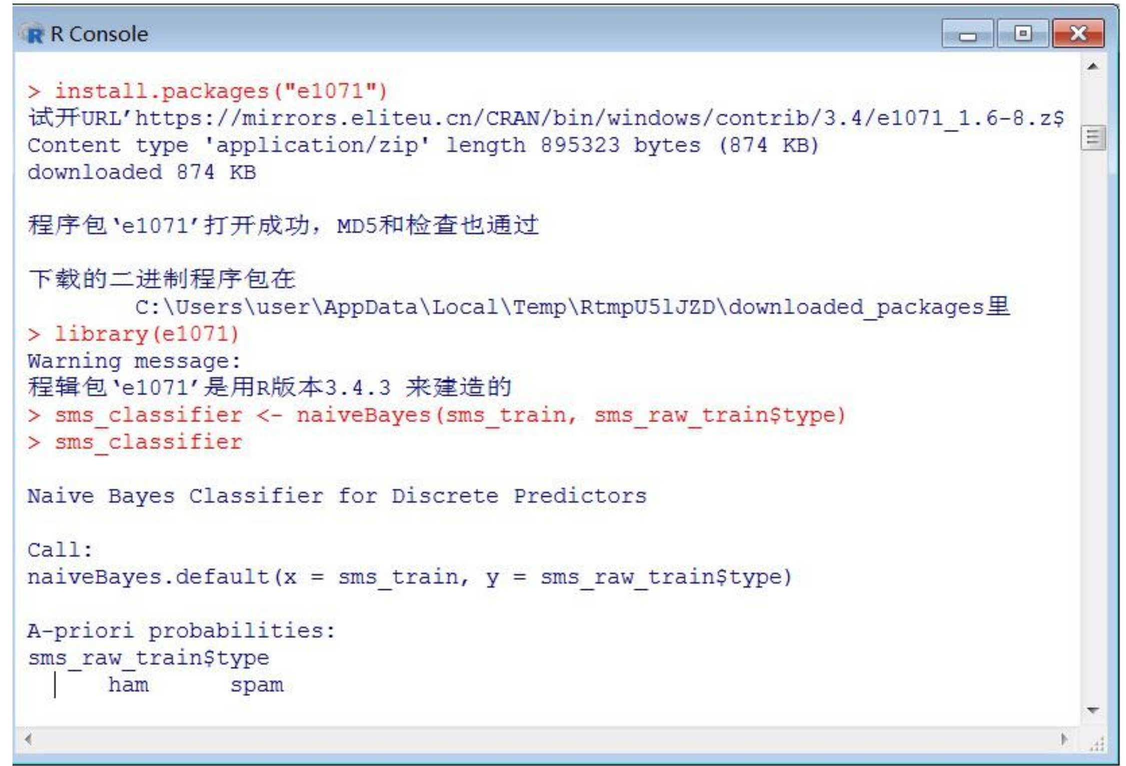


图 6.35 训练模型

（十二）评估模型性能

```
sms_test_pred <- predict(sms_classifier, sms_test)
## 如前所述，利用 predict()函数进行预测。

install.packages("gmodels")
library(gmodels)
CrossTable(sms_test_pred, sms_raw_test$type,
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
           dnn = c('predicted', 'actual'))
## 我们在这里用了三个 FALSE 来消除目前不需要的元素及其比例。
## 我们在这里使用参数 dnn(维度名称)来重新标记行和列。
```

结果

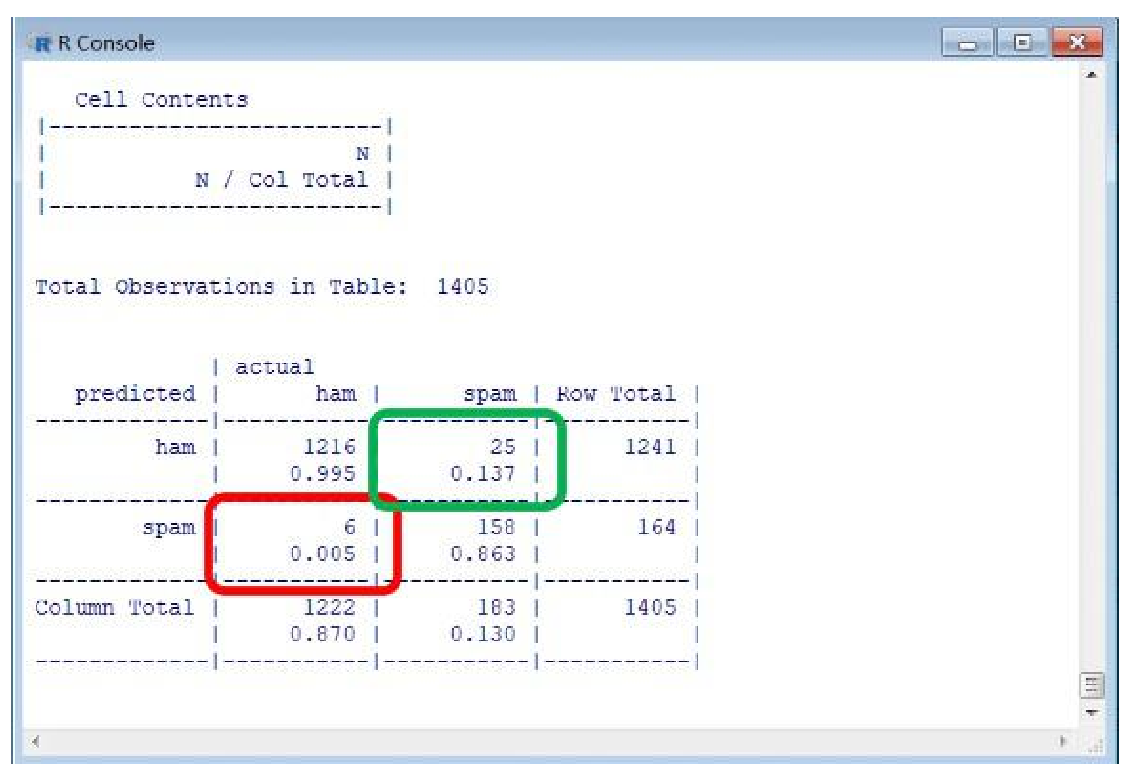
```
R Console
> install.packages("gmodels")
试开URL'https://mirrors.eliteu.cn/CRAN/bin/windows/contrib/3.4/gmodels_2.16.$
Content type 'application/zip' length 74282 bytes (72 KB)
downloaded 72 KB

程序包'gmodels'打开成功，MD5和检查也通过

下载的二进制程序包在
        C:\Users\user\AppData\Local\Temp\RtmpU5lJZD\downloaded_packages里
> sms_test_pred <- predict(sms_classifier, sms_test)
> library(gmodels)
Warning message:
程辑包'gmodels'是用R版本3.4.3 来建造的
> CrossTable(sms_test_pred, sms_raw_test$type,
+            prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
+            dnn = c('predicted', 'actual'))


   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|
```

图 6.36 评估模型性能（1）



```
   Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  1405


             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1216 |        25 |      1241 |
             |     0.995 |     0.137 |           |
-------------|-----------|-----------|-----------|
        spam |         6 |       158 |       164 |
             |     0.005 |     0.863 |           |
-------------|-----------|-----------|-----------|
Column Total |      1222 |       183 |      1405 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

图 6.37 评估模型性能（2）

如图 6.37 所示，我们可以得知以下两条重要消息：

1.有 25 条垃圾短信被错误地标注为正常短信，比例是 13.7%

2.有 6 条正常短信被错误地标注为垃圾短信，比例是 0.5%

此时，不妨先别往下看，而是问问您自己三个问题：

1.您如何判断这个结果？

2.我们需要检讨哪个结果？

3.我们需要优化哪个部分？

想清楚了之后，我们再接着往下实践。

（十三）提高模型性能

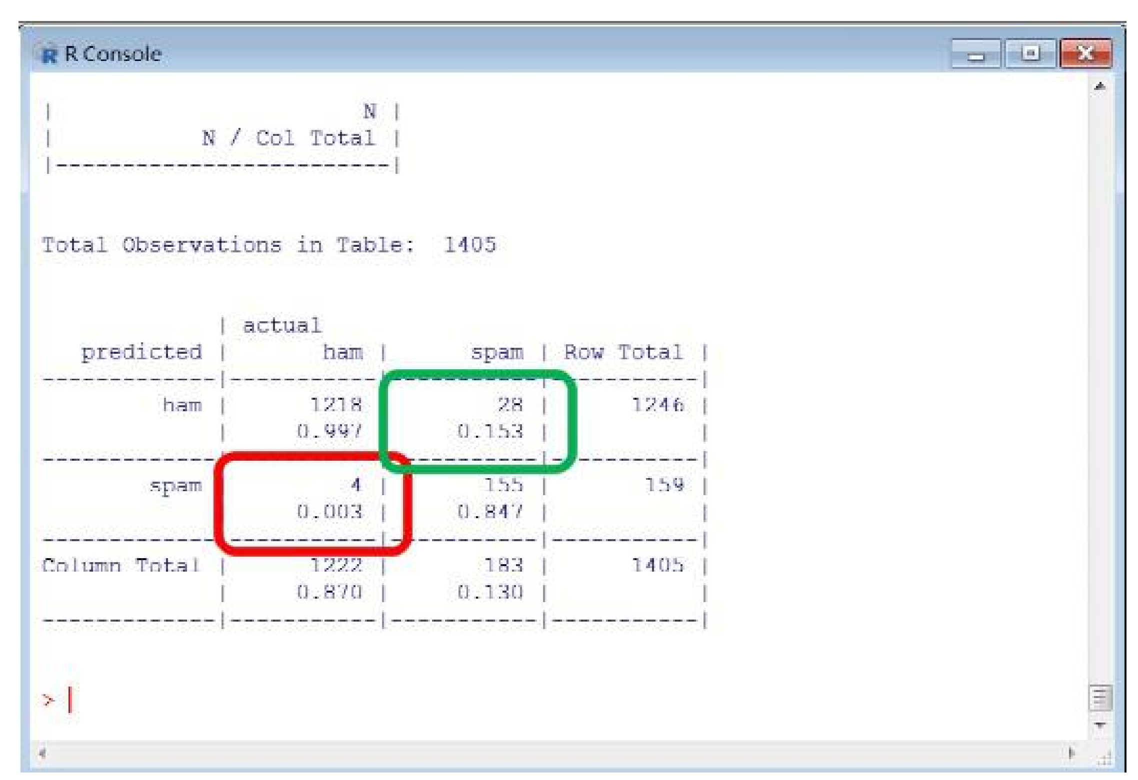
如 7.3.1.12 的图 7.37 所示，有 6 条短信被错误地标注为垃圾短信，比例是 0.5%这说明了：如果这 6 条短信中，有重要信息或者紧急信息，那么就有可能造成终端用户日后不再使用这套服务。所以，我们需要进一步分析为什么正常短信会被分为垃圾短信。

```
# 建立朴素贝叶斯模型，但是 laplace=1，使用拉普拉斯估计。
sms_classifier2 <- naiveBayes(sms_train, sms_raw_train$type, laplace = 1)

# 进行预测。
sms_test_pred2 <- predict(sms_classifier2, sms_test)

使用交叉联表（CrossTable）比较预测的分类和真实的分类。
CrossTable(sms_test_pred2, sms_raw_test$type,
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
           dnn = c('predicted', 'actual'))
```

结果



```
R Console

|                         N |
|           N / Col Total |
|-------------------------|


Total Observations in Table:  1405


             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1218 |        28 |      1246 |
             |     0.997 |     0.153 |           |
-------------|-----------|-----------|-----------|
        spam |         4 |       155 |       159 |
             |     0.003 |     0.847 |           |
-------------|-----------|-----------|-----------|
Column Total |      1222 |       183 |      1405 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|


>
```

图 6.38 提升模型性能

根据图 6.38 所示，我们得到两条重要信息：

1.正常短信被错误地标注的情况，从 6 条降到 4 条，比例从 0.5%降到 0.3%了。

2.垃圾短信被错误地标注为正常短信的情况，从 25 条上升到 28 条，比例从 13.7%上升到 15.3%了。

我们可以这么判断：

1.虽然看上去错误分类的总数增加了 1 条，但是从业务上来看，如果我们对于垃圾短信的自动过滤过于激进，那么重要信息被遗漏的可能性越大。

2.所以，我们经过了【拉普拉斯估计】之后，所改造的【朴素贝叶斯】模型，其性能提升了。

然而，我们还需要进一步理解【朴素贝叶斯】才能了解为什么模型优化了。我们总结一下目前为止的练习背后，所需要理解的知识点：

1.估计，依赖概率方法或者描述不确定性的方法，通过过去发生的事件的数据，来推断未来的事件。

2.相关事件是建立预测模型的基础。

3.联合概率可以通过贝叶斯定理的公式进行计算，因为估计一个结果的概率，从众多属性中提取的信息要被考虑。

4.然而，贝叶斯公式的计算很复杂，而朴素贝叶斯用一个事件相互独立的简单假设，得到可以用于海量数据集的简化版贝叶斯算法。

5.朴素贝叶斯分类器经常被用在文本文件的分类上，然而，它需要事前准备文本数据、预处理文本，以及文本可视化。

6.朴素贝叶斯往往也要考虑拉普拉斯估计的问题，避免某一证据的概率为零的情况下，抵消或者否决其他所有证据。

7.朴素贝叶斯有很多不合理的假设，那么它的结果是否能够应用？

对于上述 7 的问题，我们多半从数据科学的工程视角，而不是科学视角，来进行思考和

回答（参考第一章第一节的内容）。

尽管朴素贝叶斯的假设在大多数情况下很难成立，然而在大多数情况下，朴素贝叶斯算法却在分类学习任务中具有很好的应用效果；这种结果的其中一种解释是：只要预测的分类值是准确的，那么是否获得精确的概率估计并不重要，例如，垃圾邮件过滤器如果能够正确识别垃圾邮件，那么它的预测具有 51%的准确率还是 99%的准确率，对于完成分类任务而言就不是那么重要了。

至此，朴素贝叶斯的练习结束了，但我们除了掌握怎么使用 R 进行朴素贝叶斯的机器学习，更重要的是通过练习来理解有监督学习的内涵。因此，请您试着思考以下几个问题：

1.为什么我们要从数据找模型，而不是从模型看数据？

2.为什么我们要从研究问题去找算法，而不是从算法考虑问题？

3.为什么我们要从业务经验进行判断，而不是数学模型进行判断？

我们看待数据科学，最重要的是用科学去思考数据，而不是用数据去解释科学。所以，任何练习之后，都不要停止思考，把这个习惯养成之后，带到科学研究或者日常生活里，都大有助益。

二、最近邻学习

（一）最近邻的特性

最近邻 kNN 算法（k-Nearest Neighbors）又被称为“懒惰学习”（lazy learning）、基于实例的学习（instance-based learning）或者机械性学习（rote learning）。
听名字就知道，它不是真正进行机器学习（追求最优参数的解）而是优化存储训练用的数据（暴力破解），能够提高训练过程的效率，但是预测过程相对缓慢。
往往单纯而且暴力的方法，可以快速有效处理问题，但不一定是最好的处理方法；最近邻能够满足自动化需求但不足以建立理论。

（二）最近邻的场景和方法

场景：当理论概念难以定义（特征和类别之间的关系错综复杂），但是人们明确知道它是什么（有些对象非常相似）时，就把具有具有相似标记的对象归类。

方法：将对象视为一个多维特征空间（feature space）内的坐标上的一个点，计算距离函数（distance function）或者两者之间相似性。

邻居数量 k 是偏差-方差权衡（bias-variance tradeoff）的结果，它决定把当前从数据得到的模型推广到未来其他数据上的优劣。

（三）最近邻的核心问题：K 值设置

关于 k 值的设置：
1.k=训练集数量的平方根。
2.测试多个 k 值的结果。
3.选择较大 k 值，同时进行权重投票（weighted voting）：近邻者的权重大于较远者。

（四）最近邻的细节：虚拟变量

虚拟变量（Dummy coding）：
1.1 表示一个类别，0 表示其他类别。
2.具有 n 个类别的特征（名称变量 name variable），可用（n-1）个层级，创建二元的指

标变量（下标变量 indexed variable），进行虚拟变量的编码。
如果名称变量是有序组成，并且类别之间的距离相等，则可给予每个名称变量的类别进行编号，再用 min-max 标准化。

（五）最近邻的细节：标准化

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

图 6.39 min-max 标准化

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \text{Mean}(X)}{\text{StdDev}(X)}$$

图 6.40 z-score 标准化

（六）数据选择及其背景

借用来自 UCI Machine Learning Repository 的 Cancer 数据集，进行演示和讨论（http://archive.ics.uci.edu/ml）。 所讨论的实例是一组经过整理后的临床数据，医生在显微镜下检查可疑的肿块，记录肿块的细胞核的 10 种特征，以及肿块属于良性（B）或者恶性（M）肿瘤。目的是能够找到合适模型，用来节省医生诊断时间，而增加医生对病患的治疗时间；并且形成自动化筛选系统，降低人为主观判断病情的可能。

```
#导入数据集。
wbcd <- read.csv("wisc_bc_data.csv", stringsAsFactors = FALSE)

#查看数据结构。
str(wbcd)

#剔除掉 ID 特征（否则 ID 被用作“预测”后，会造成模型过度拟合）
wbcd <- wbcd[-1]
```

结果：

```
'data.frame':	569 obs. of  32 variables:
 $ id                     : int  842302 842517 84300903 84348301 84358402 843786 84$
 $ diagnosis              : chr  "M" "M" "M" "M" ...
 $ radius_mean            : num  18 20.6 19.7 11.4 20.3 ...
 $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 ...
 $ perimeter_mean         : num  122.8 132.9 130 77.6 135.1 ...
 $ area_mean              : num  1001 1326 1203 386 1297 ...
 $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
 $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
 $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
 $ concave.points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
 $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
 $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
 $ radius_se              : num  1.095 0.543 0.746 0.496 0.757 ...
 $ texture_se             : num  0.905 0.734 0.787 1.156 0.781 ...
 $ perimeter_se           : num  8.59 3.4 4.58 3.44 5.44 ...
 $ area_se                : num  153.4 74.1 94 27.2 94.4 ...
 $ smoothness_se          : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
 $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
 $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
 $ concave.points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
 $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
 $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
```

图 6.41 肿瘤数据集

（七）数据检查

按照惯例，拿到数据，直接检查。不过，此处我们要做的是编辑【变量属性】为因子变量，这样可以进行之后的分析。以及，判断是否需要进一步【编辑数据】。

```
#查看变量 dianosis（良性或者恶性肿瘤），该变量是预测结果。
table(wbcd$diagnosis)

#编辑变量 dianosis 的属性为因子（factor），赋予标签说明。
wbcd$diagnosis <- factor(wbcd$diagnosis, levels = c("B", "M"), labels = c("Benign",
"Malignant"))

round(prop.table(table(wbcd$diagnosis)) * 100,  digits  =  1)

#  观察三个变量的特征（均是数值型）。
summary(wbcd[c("radius_mean",  "area_mean",  "smoothness_mean")])
```

结果：

```
> wbcd$diagnosis <- factor(wbcd$diagnosis, levels = c("B", "M"),
+                          labels = c("Benign", "Malignant"))
> round(prop.table(table(wbcd$diagnosis)) * 100, digits = 1)

   Benign Malignant 
     62.7      37.3 
> summary(wbcd[c("radius_mean", "area_mean", "smoothness_mean")])
  radius_mean       area_mean      smoothness_mean  
 Min.   : 6.981   Min.   : 143.5   Min.   :0.05263  
 1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637  
 Median :13.370   Median : 551.1   Median :0.09587  
 Mean   :14.127   Mean   : 654.9   Mean   :0.09636  
 3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530  
 Max.   :28.110   Max.   :2501.0   Max.   :0.16340  
> |
```

图 6.42 最近邻之前的数据检查

由于 kNN 的距离计算，在很大程度上依赖【输入特征】的【测量尺度】，由于 smoothness 在 0.05 到 0.16 之间，而 area 在 143.5 到 2501.0 之间，所以如果进行 kNN 的【距离计算】，则 area-mean 的影响就比 smoothness-mean 高出很多，从而潜在干扰结果。

所以，我们需要针对 Min 和 Max 采用【标准化】方法，进行调整。

（八）标准化

根据图 6.39 的标准化公式，我们在 R 可以如下实现：

```
# 标准化数值型数据，在 R 里建构我们的 normalize 函数。

normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x)))
}

#函数 normalize()已被建立，可用几个向量测试。
normalize(c(1, 2, 3, 4, 5))
normalize(c(10, 20, 30, 40, 50))
```

因为标准化的函数已经写好如上，所以我们执行标准化的时候，就只一行而已。

```
wbcd_n <- as.data.frame(lapply(wbcd[2:31], normalize))
summary(wbcd_n$area_mean)

## 函数 as.data.frame 把函数 lapply 返回【列表】转换成【数据框】。
## 函数 lapply()可以输入【列表】然后把任意【函数】应用到每个列表中的【元素】，因为【数据框】的向量等长，所以可以把【函数 normalize()】应用到 lappy()列表的每个【元素】上。
## 上述指令，把 normalize()函数应用到数据框 wbcd 的第 2 到 31 列，又把产生的结果从列表转换成为数据框。下述指令，予以确认。
```

结果：

```
> normalize <- function(x) {
+   return ((x - min(x)) / (max(x) - min(x)))
+ }
> normalize(c(1, 2, 3, 4, 5))
[1] 0.00 0.25 0.50 0.75 1.00
> normalize(c(10, 20, 30, 40, 50))
[1] 0.00 0.25 0.50 0.75 1.00
> wbcd_n <- as.data.frame(lapply(wbcd[2:31], normalize))
> summary(wbcd_n$area_mean)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.1174  0.1729  0.2169  0.2711  1.0000
> |
```

图 6.43 标准化

（九）准备数据：创建【训练集】和【测试集】

不用多说，直接上代码，因为准备机器学习了，而且是有监督学习。

```
wbcd_train <- wbcd_n[1:469, ]
wbcd_test <- wbcd_n[470:569, ]
# 从数据框提取数据的语法是[行，列]，如果没有填写（如上），则表示所有的行或者列都被包括在内。上述脚本，抽取了所有列。

# 创建带有变量 diagnosis 并且作为【因子】的向量。
wbcd_train_labels <- wbcd[1:469, 1]
wbcd_test_labels <- wbcd[470:569, 1]
```

（十）训练模型

加载 class 包，因为其中包括了函数 kNN()可实现最近邻算法。

```
library(class)
##使用函数 knn()对测试集进行分类，就每条数据返回一个预测标签，形成一个因子向量。

wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,cl = wbcd_train_labels, k=21)
## 为了避免票数相等的情况，选择训练集（数据 469 条）数据的平方根的奇数（21）作为 k 参数，指定投票中所包括的邻居数量。
```

稍微对于 class 包，做点说明，便于日后继续使用。程序包 class 的函数 kNN()的解释：

1.train：具有数值型数据的数据框，用于机器学习的训练集。

2.test：具有数值型数据的数据框，用于机器学习的测试集。

3.class：具有训练集的每一行分类的因子向量。

4.k：最近邻的数量。

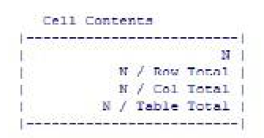
所以，函数 knn()应当返回一个因子向量，其中包括每条数据所进行分类预测的标签。

为了得知 wbcd_test_pred 的预测分类与 wbcd_test_labels 向量的已知类别的匹配程度；加载 gmodels 包，使用函数 CrossTable()进行两个向量之间一致性的交叉表。

```
library(gmodels)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
  prop.chisq=FALSE)
#指定参数 prop.chisq=FALSE 则会从输出结果中去除不必要的卡方值（chi-square）。
```

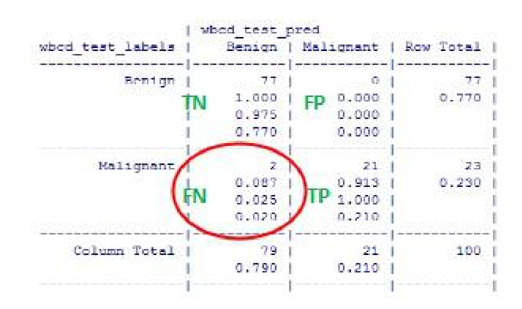
（十一）评估模型性能

如果运行 7.3.2.10 的代码之后，会有如图 7.44 的结果



```
  Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|
```

Total Observations in Table: 100



wbcd_test_labels	wbcd_test_pred Benign	Malignant	Row Total
Benign	77	0	77
	1.000	0.000	0.770
	0.975	0.000	
	0.770	0.000	
Malignant	2	21	23
	0.087	0.913	0.230
	0.025	1.000	
	0.020	0.210	
Column Total	79	21	100
	0.790	0.210	

图 6.44 评估模型性能（1）

尽管刚刚 R 的 kNN 计算当中，得到 98%的准确率，然而：

在 100 个肿块当中，有 2 个被错误分类，如果庸医凭借这种假阴性的结果，就判断出这是良性肿瘤（而事实上是恶性肿瘤），这会让病人以为自己没有癌症，从而可能导致疾病继续蔓延。

因此，就此情况而言，我们就还有两项任务：

1.为了避免假阴性（FN），增加假阳性（FP），牺牲准确率。

2.仔细再做检查，这依赖专家知识、经验和道德。

（十二）提高模型性能

之前采用的 min-max 标准化方式，设定了最大最小值，如果数据集里面具有极端数值（异常值）会往中心压缩；如果没有这种预设，则异常值在距离计算之中，会占有更大的权重。

不预设最大最小值，可以使用 R 内置的函数 scale()进行 z-score 标准化，该函数可以直接处理数据框，不必使用函数 lapply()。

```
wbcd_z <- as.data.frame(scale(wbcd[-1]))
# z 是为了表明进行 z-score 标准化。
# -1 是为了去掉 ID。
```

结果：

```
> wbcd_z <- as.data.frame(scale(wbcd[-1]))
> summary(wbcd_z$area_mean)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.4530  -0.6666  -0.2949   0.0000   0.3632   5.2460
```

图 6.45 提高最近邻的模型性能（1）

当 z-score 标准化之后，均值应当是 0，而且分布情况（最大最小值、四分位数、均值等的距离）应当十分紧凑。

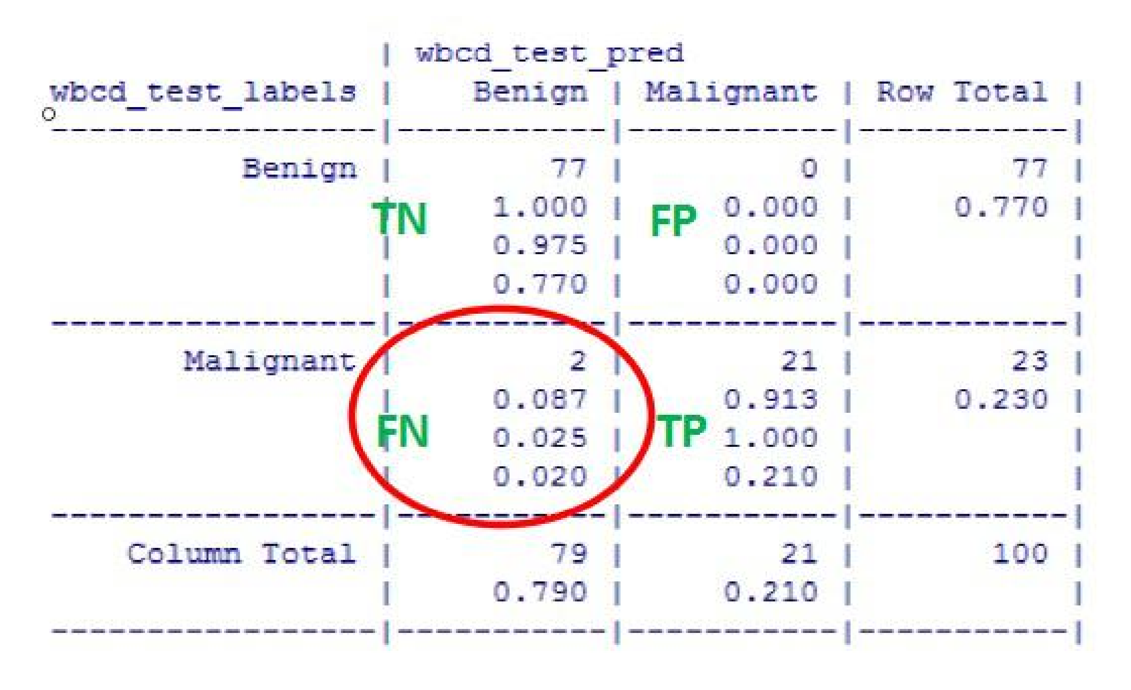
大约 3 或者小于-3 的数值，在 z-score 标准化之中，表示的是异常值（极其罕见与其他多数不一样的数值）。

目前情况合理。

另外，与之前建模方式相同，建立训练集和测试集，进行分类，以及比较预测结果和实际结果。

```
wbcd_train <- wbcd_z[1:469, ]
wbcd_test <- wbcd_z[470:569, ]
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=21)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,  prop.chisq=FALSE)
```

结果：



	wbcd_test_pred		
wbcd_test_labels	Benign	Malignant	Row Total
Benign	77 1.000 0.975 0.770	0 0.000 0.000 0.000	77 0.770
Malignant	2 0.087 0.025 0.020	21 0.913 1.000 0.210	23 0.230
Column Total	79 0.790	21 0.210	100

图 6.46 提高最近邻的模型性能（2）

预测情况与之前模型大致相同：（1）准确率 98%的效果；（2）存在 FN 为 2 的情况。

此例，采用 Min-Max 标准化和 z-score 标准化的结果差别不大；这批数据中的异常值对以 kNN 进行分类的结果影响有限。

因此，在不调整训练集和测试集的比例的情况下，尝试调整参数 k 进行测试和结果比较。

```
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=5)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=11)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=15)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=21)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=27)
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
```

结果：

表 6.3 K 折交叉验证

k 值	假阴性 FN	假阳性 FP	准确率
1	2	5	93
5	0	3	97
11	1	1	98
15	2	0	98
21	2	0	98
27	2	0	98

不妨考虑看看，选择哪个 k 值作为参数？

提示：

1.为了避免假阴性（FN），增加假阳性（FP），牺牲准确率。

2.学科领域的问题，仍然依赖专家知识、经验和道德。

此外，补充一句话：

如果是另外一批受测者，可能发现的最优模型也会不同。

三、决策树

决策树，顾名思义，是为了【决策】进行判断的方法。它要求在众多变量当中，优先考虑哪个变量，通过这个变量的结果，再考虑哪个变量，一直到能够确定最终答案为止，这整个路径，看起来就像树枝状的路径，从主干（数据集的整体）通过树枝（决策树模型）一直延伸到末端的叶子（数据集的某个单个样本）的好坏（判断结果）上。

请先安装 mboost 程序包（Model-based Gradient Boosting）。运行 mboost 包之前，需要安装 parallel 和 stabs 包（mboost 需要调用这两个程序）。

```
library(mboost)。
data("bodyfat", package = "TH.data")
str(bodyfat)
head(bodyfat)
```

结果：

```
R Console
> library(mboost)
> ?mboost
starting httpd help server ... done
> data("bodyfat", package = "TH.data")
> str(bodyfat)
'data.frame':   71 obs. of  10 variables:
 $ age         : num  57 65 59 58 60 61 56 60 58 62 ...
 $ DEXfat      : num  41.7 43.3 35.4 22.8 36.4 ...
 $ waistcirc   : num  100 99.5 96 72 89.5 83.5 81 89 80 79 ...
 $ hipcirc     : num  112 116.5 108.5 96.5 100.5 ...
 $ elbowbreadth: num  7.1 6.5 6.2 6.1 7.1 6.5 6.9 6.2 6.4 7 ...
 $ kneebreadth : num  9.4 8.9 8.9 9.2 10 8.8 8.9 8.5 8.8 8.8 ...
 $ anthro3a    : num  4.42 4.63 4.12 4.03 4.24 3.55 4.14 4.04 3.91 3.66 ...
 $ anthro3b    : num  4.95 5.01 4.74 4.48 4.68 4.06 4.52 4.7 4.32 4.21 ...
 $ anthro3c    : num  4.5 4.48 4.6 3.91 4.15 3.64 4.31 4.47 3.47 3.6 ...
 $ anthro4     : num  6.13 6.37 5.82 5.66 5.91 5.14 5.69 5.7 5.49 5.25 ...
> head(bodyfat)
   age DEXfat waistcirc hipcirc elbowbreadth kneebreadth anthro3a anthro3b
47  57  41.68     100.0   112.0          7.1         9.4     4.42     4.95
48  65  43.29      99.5   116.5          6.5         8.9     4.63     5.01
49  59  35.41      96.0   108.5          6.2         8.9     4.12     4.74
50  58  22.79      72.0    96.5          6.1         9.2     4.03     4.48
51  60  36.42      89.5   100.5          7.1        10.0     4.24     4.68
52  61  24.13      83.5    97.0          6.5         8.8     3.55     4.06
   anthro3c anthro4
47     4.50    6.13
48     4.48    6.37
```

图 6.47 决策树（1）：肿瘤数据的基本情况

作为一个数据框，共有 71 个对象（客户信息）和 10 个变量（测量数值）。名为 bodyfat 的数据集，由 mboost 包提供，变量名称以及含义，如下：

- age：年龄
- DEXfat：以 DEX 计算身体脂肪重量
- waistcirc：腰围
- hipcire：臀宽
- elbowbreadth：肘宽
- kneebreadth：膝宽
- anthro3a：三项人体测量的对数和
- anthro3b：三项人体测量的对数和
- anthro3c：三项人体测量的对数和
- anthro4：三项人体测量的对数和

（一）种树

```
#按照前面步骤，分别安装如下程序包
install.packages("party")
##注解：会自动加载 grid, mvtnorm, modeltools, stats4, strucchange, zoo, sandwich
等程序包（都封装在一起了）。
install.packages("rpart")

# 加载 rpart 包，加载 mboost 并且调用 bodyfat 数据集。
library(rpart)
library(mboost)
data("bodyfat", package = "TH.data")
attributes(bodyfat)

# 将 bodyfat 分解为训练集和测试集
set.seed(1234)
id<- sample(2, nrow(bodyfat), replace=TRUE, prob=c(0.65, 0.35))
trainingdataset <- bodyfat[id==1,]
testingdataset <- bodyfat[id==2,]

## 赋值 id 为两个样本（sample）；将 iris 数据集，也就是这里的 nihao（你好）数据集
划分为两个子集：训练集和测试集。其中 65%的数据用于训练，而 35%的数据用于测试。

#在训练集 trainingdataset 上，建立决策树：
myTree <- DEXfat ~ waistcirc + hipcirc + elbowbreadth + kneebreadth
bodyfat_rpart <- rpart(myTree, data=trainingdataset, control =
rpart.control(minsplit = 10))
##这里 rpart.control 意指：对决策树（bodyfat_rpart）进行设置。
##这里 minsplit 意指：最小分支节点数。其他包括：子节点最小样本数（minbucket）、树
的深度（maxdepth）等。
## 建立决策树，命名为 myTree 的向量空间，指定了 DEXfat 为目标变量（应变量），其他
为自变量。

##输出并且查看决策树的属性（行名、列名、类）。
attributes(bodyfat_rpart)
```

此处键入 set.seed 是为了生成随机数，在（）之中的数字，没有特别意义，但是每次实现方法的时候，可以得到相同随机数。因此，研究结论是不应该建立在某次单一的随机数的基础上（否则九失去随机数的意义了），但是其他研究人员可以据此验证这一次的结果（否则人们无从知道结果不一致的原因）。

结果：

```
R Console

> set.seed(1234)
> id<- sample(2, nrow(bodyfat), replace=TRUE, prob=c(0.65, 0.35))
> trainingdataset <- bodyfat[id==1,]
> testingdataset <- bodyfat[id==2,]
> myTree <- DEXfat ~ waistcirc + hipcirc + elbowbreadth + kneebreadth
> bodyfat_rpart <- rpart(myTree, data=trainingdataset, control = rpart.contr$
> attributes(bodyfat_rpart)
$names
 [1] "frame"             "where"             "call"
 [4] "terms"             "cptable"           "method"
 [7] "parms"             "control"           "functions"
[10] "numresp"           "splits"            "variable.importance"
[13] "y"                 "ordered"

$xlevels
named list()

$class
[1] "rpart"

> |
```

图 6.48 决策树（2）：属性

以上，可以注意几个信息，包括：

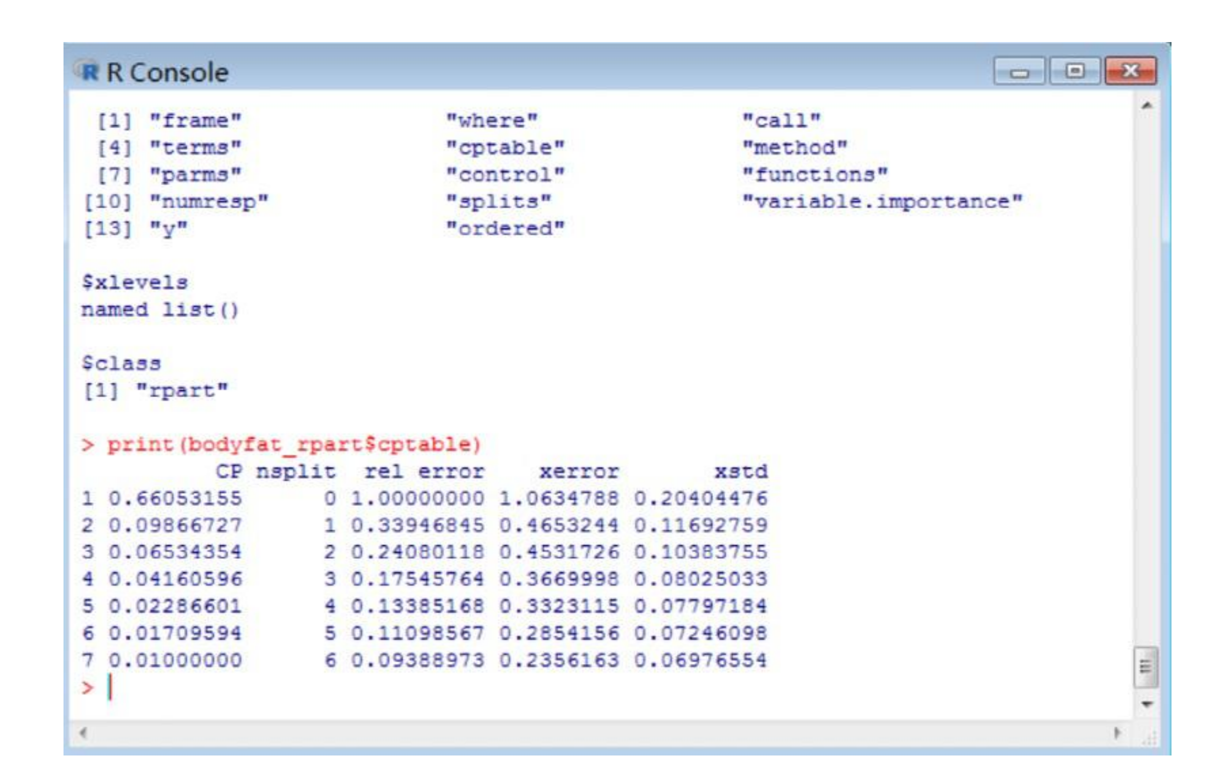
- method：对应决策树末端数据类型的变量分割方式：连续型 anova、离散型 class、计数型 poisson、生存分析型 exp 等。
- parms：先验概率、损失矩阵、分类纯度的度量方法的参数设置。
- cptable（complexity parameter table）：点的复杂度，决定这棵决策树是否需要修剪的重要参考指标。

以下，关注决策树的本质，即：属性和参数

```
#输出 bodyfat_rpart（决策树）的 cptable 属性（复杂参数）。
print(bodyfat_rpart$cptable)

##“nsplit”意指“折数”；
##“xerror”列，意指“交叉验证的估计误差”；
##“xstd”列，是指“标准误差”；
## 平均相对误差 = xerror ± xstd
```

结果：


```
 [1] "frame"            "where"            "call"
 [4] "terms"            "cptable"          "method"
 [7] "parms"            "control"          "functions"
[10] "numresp"          "splits"           "variable.importance"
[13] "y"                "ordered"

$xlevels
named list()

$class
[1] "rpart"

> print(bodyfat_rpart$cptable)
          CP nsplit  rel error    xerror       xstd
1 0.66053155      0 1.00000000 1.0634788 0.20404476
2 0.09866727      1 0.33946845 0.4653244 0.11692759
3 0.06534354      2 0.24080118 0.4531726 0.10383755
4 0.04160596      3 0.17545764 0.3669998 0.08025033
5 0.02286601      4 0.13385168 0.3323115 0.07797184
6 0.01709594      5 0.11098567 0.2854156 0.07246098
7 0.01000000      6 0.09388973 0.2356163 0.06976554
>
```

图 6.49 决策树（3）：属性的复杂参数的部分

（二）输出决策树规则。

```
# 输出规则，并且绘制决策树，进行解读、解释、检查。
print(bodyfat_rpart)

## split 折，分裂到每一层的规则；
## n 数量；
## deviance 偏差值；
## yval*denotes terminal node 变量 Y 的数值代表终端节点。
```

结果：

```
 [1] "frame"              "where"              "call"
 [4] "terms"              "cptable"            "method"
 [7] "parms"              "control"            "functions"
[10] "numresp"            "splits"             "variable.importance"
[13] "y"                  "ordered"

$xlevels
named list()

$class
[1] "rpart"

> print(bodyfat_rpart$cptable)
          CP nsplit  rel error    xerror       xstd
1 0.66053155      0 1.00000000 1.0634788 0.20404476
2 0.09866727      1 0.33946845 0.4653244 0.11692759
3 0.06534354      2 0.24080118 0.4531726 0.10383755
4 0.04160596      3 0.17545764 0.3669998 0.08025033
5 0.02286601      4 0.13385168 0.3323115 0.07797184
6 0.01709594      5 0.11098567 0.2854156 0.07246098
7 0.01000000      6 0.09388973 0.2356163 0.06976554
>
```

图 6.50 决策树（4）：规则

```
#绘制决策树，并在决策树的图像上增加文本。
plot(bodyfat_rpart)
text(bodyfat_rpart, use.n=T)
```

结果：

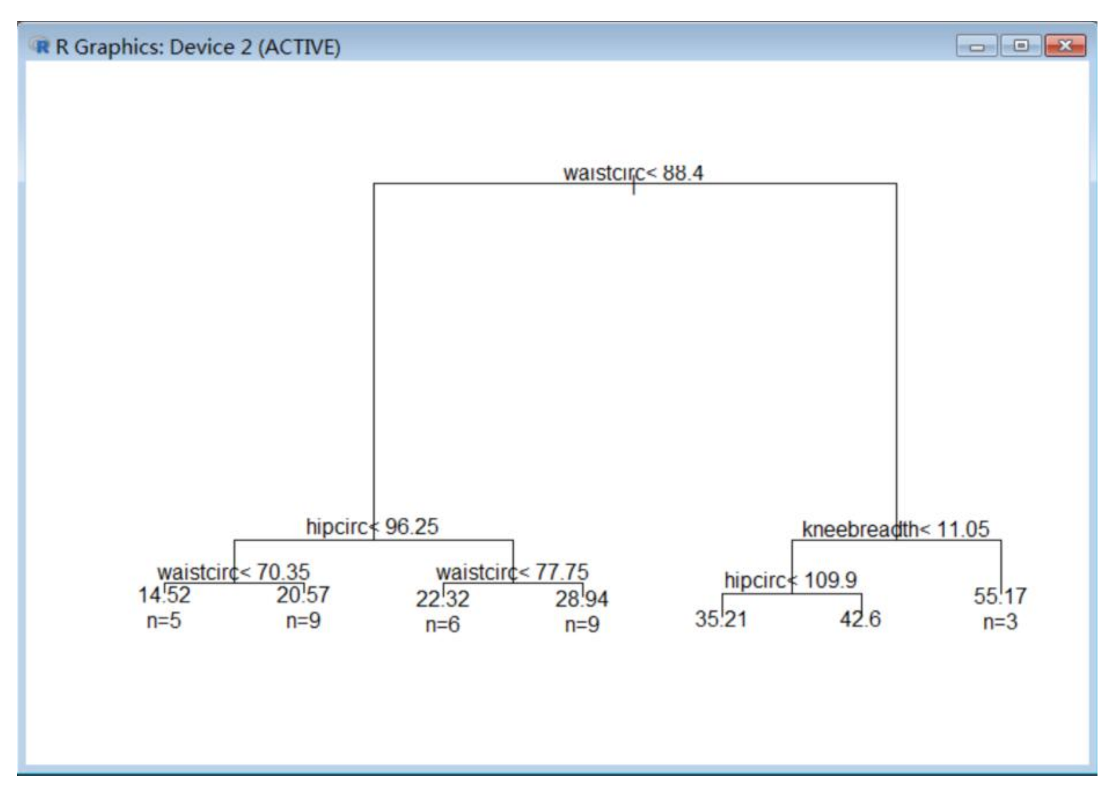


图 6.51 决策树（5）：树干（决策过程）

（三）选择具有最小误差的决策树。

```
select <- which.min(bodyfat_rpart$cptable[,])
##使用 which 条件函数，获取 bodyfat_rpart 决策树中的 cptable 复杂参数的 xerror 的最小值的行索引。
###将最小的 xerror 值的行，赋予 select

select02 <- bodyfat_rpart$cptable[select,"CP"]
#获取最小值行的 CP 列值;
#也就是获取(获取 bodyfat_rpart 决策树中的 cptable 复杂参数的 xerror 的最小值的行索引）的 CP 列值;
##将上述 CP 列值，赋予 select02

bodyfat_prune <- prune(bodyfat_rpart, cp = select02)
#对初始树进行修剪;
#根据“复杂参数”cp 修剪掉 bodyfat_rpart 递归中最不重要的分支，以获取一个最优化的决策树枝干。
##将上述修剪后的决策树，赋予 bodyfat_prune

print(bodyfat_prune)
#输出 bodyfat_prune 修剪后的决策树的规则。

plot(bodyfat_prune)
text(bodyfat_prune, use.n=T)
#绘制修剪后的决策树。
```

结果：

```
R Console
> select02 <- bodyfat_rpart$cptable[select,"CP"]
> bodyfat_prune <- prune(bodyfat_rpart, cp = select02)
> print(bodyfat_prune)
n= 53

node), split, n, deviance, yval
      * denotes terminal node

 1) root 53 6882.30700 30.91283
   2) waistcirc< 88.4 29  958.36990 22.48759
     4) hipcirc< 96.25 14  222.26480 18.41143
       8) waistcirc< 70.35 5   28.84808 14.52200 *
       9) waistcirc>=70.35 9   75.75716 20.57222 *
     5) hipcirc>=96.25 15  286.39080 26.29200
      10) waistcirc< 77.75 6   30.73455 22.32500 *
      11) waistcirc>=77.75 9   98.28540 28.93667 *
   3) waistcirc>=88.4 24 1377.95600 41.09333
     6) kneebreadth< 11.05 21  467.16460 39.08286
      12) hipcirc< 109.9 10  131.66640 35.21000 *
      13) hipcirc>=109.9 11   49.15325 42.60364 *
     7) kneebreadth>=11.05 3  231.73310 55.16667 *
>
```

图 6.52 决策树（6）：修剪后的决策树的规则

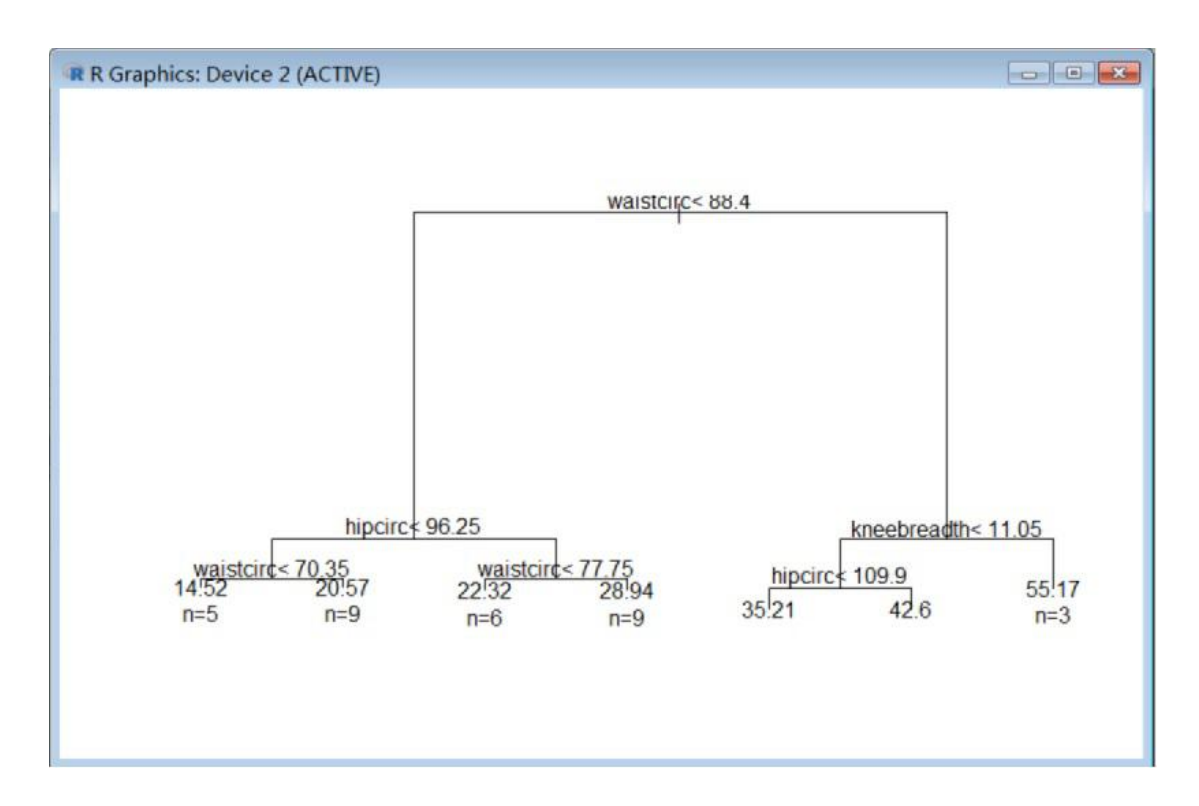


图 6.53 决策树（7）：修建过的决策树的树干（决策过程）

（四）将预测结果与真实数据进行对比

```
# 使用测试集，对已经建好的决策树，进行测试。

bodyfat_test <- predict(bodyfat_prune, newdata = testingdataset)
## 使用 predict 函数，在已经建立的决策树 bodyfat_prune 的基础上，导入测试数据集 testingdataset 进行预测。
## 将预测结果，赋予 bodyfat_test

xlim <- range (bodyfat$DEXfat)
## range 意指：最小值和最大值的范围区域。
## 获取原始数据值 bodyfat 的 DEXfat 变量的最小值和最大值范围。
## 将最大最小的范围区间，赋予 xlim

plot(bodyfat_test ~ DEXfat, data = testingdataset, xlab= "观测值", ylab="预测值", xlim=xlim, ylim=xlim)

# plot 意指：绘制散点图。
## 此行代码旨在绘制预测值与实际值的散点图，以说明预测准确性。
## xlab 和 ylab 意指：在 x 轴和 y 轴上的标签（Label 简写 lab）；
## xlim 和 ylim 意指：x 轴和 y 轴长度，这里把最大最小区间 xlim 赋予。

abline (a=0, b=1)
## 在该图上添加截距为 0 而斜率为 1 的直线，直观说明预测是否准确。
```

结果：

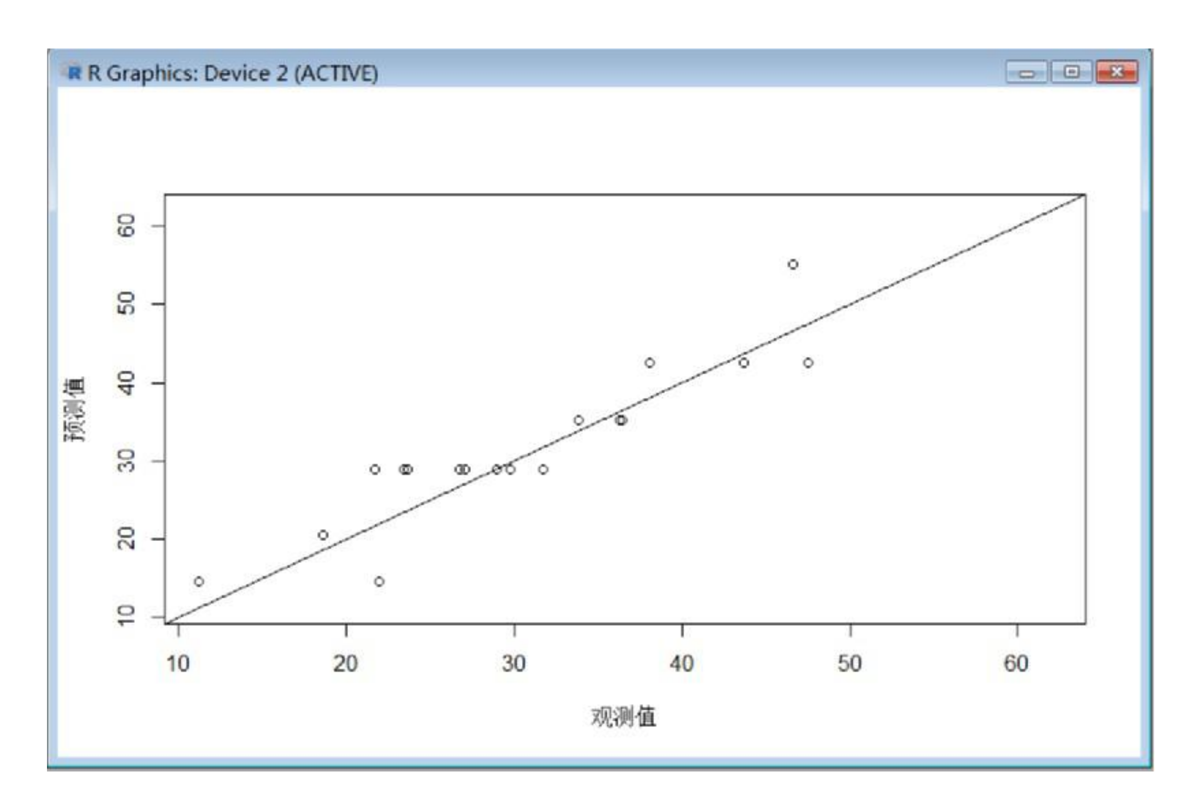


图 6.54 决策树（8）：预测结果与真实数据的对比

我们永远不会真正理解学习，直到我们能够制造出学习许多不同东西的机器，经过多年，随着时间推移成为更好的学习者。（Tom M. Mitchell. 2014）[1]

（五）模型评估

```
bodyfat_test
#查看 bodyfat_test 里的各个数值。
head(bodyfat[1:3,])
#查看原始数据 bodyfat 里的前三行数据。
```

我们发现起始数据是第 47 行；所以，接下来查看数据时，要从第 47 行开始（减去 47）

```
bodyfat[c(51,55,57,60,62,72,74,75)-46,]$DEXfat
bodyfat[c(82,85,86,93,96,99,104,106)-46,]$DEXfat
bodyfat[c(107,112)-46,]$DEXfat
#查看原始数据 bodyfat 里的 DEXfat 变量的列，并且对应 bodyfat_test 的行，输出数据。
```

结果：

[1] http://cs.brown.edu/events/talks/mitchell.tom.html

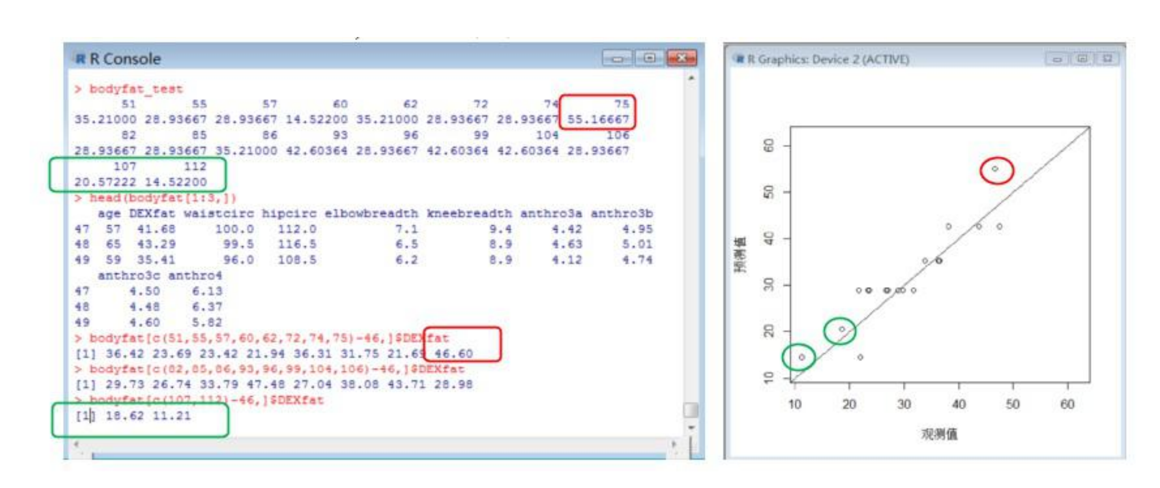


图 6.55 决策树（9）：预测与真实对照的数据与图形的对照

如果我们查看图 6.55 的两组数据，并且在散点图上（把图 6.54 不变的情况下，只是把图框拉直为正方形而不是长方形），找到坐标。就会发现这两组数据是偏离其他数据点。

这么看来，如果单看图 6.54 和图 6.55 会得到两种不同的直观判断，因此，我们应该还需要数值判断，此时可以借用皮尔森相关系数。

```
v <-bodyfat[c(51,55,57,60,62,72,74,75,82,85,86,93,
    + 96,99,104,106,107,112)-46,]$DEXfat
w <-bodyfat_test
cor.test (v,w)
```

结果：

```
R Console
> cov(bodyfat[c(51,55,57,60,62,72,74,75,82,85,86,93,96,99,104,106,107,112)-4$
[1] 90.4961
> cor(bodyfat[c(51,55,57,60,62,72,74,75,82,85,86,93,96,99,104,106,107,112)-4$
[1] 0.9133464
> v <-bodyfat[c(51,55,57,60,62,72,74,75,82,85,86,93,96,99,104,106,107,112)-4$
> w <-bodyfat_test
> cor(v,w)
[1] 0.9133464
> cor.test (v,w)

        Pearson's product-moment correlation

data:  v and w
t = 8.9723, df = 16, p-value = 1.215e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7783950 0.9676128
sample estimates:
      cor
0.9133464

>
```

图 6.56 决策树（10）：模型评估的相关系数的分析结果

根据图 6.56 所示，相关系数 0.9133464 表示高度正相关。如果纯粹用图，只能凸显重点，如果以数值计算的方式，则较为能够判断准确和肯定。当然，前提是符合数据科学的主

要原则，即：以科学解释或者分析数据，而非以数据进行科学方式的解释。

（六）过拟合的讨论

以上，我们完成了八个步骤，实现了初步的决策树的机器学习。

1. 获得数据集。
2.初步查看数据集的内容。
3.将数据集分为训练集和测试集。
4.利用训练集，建立决策树。
5.选择最小误差的决策树。
6.利用测试集和决策树，进行预测。
7.将预测结果与真实数据进行对比。
8.判断模型是否合理。

然而，我们还需要进一步考虑模型是否【过度拟合】（过拟合）的情况。此时，有两个重要概念：

1.拟合（fit），意指：把平面上一系列的点，用一条光滑的曲线连接起来。因为这条曲线有无数种可能，从而有各种拟合方法。该曲线可以用函数表示，不同函数有不同命名。
2.过度拟合（over-fitting），意指：模型能够正确分类样本数据（训练集中的样本数据），但是对于训练集以外的数据却不能够正确分类。

过拟合的原因有两方面：

一是模型（算法）的问题，模型太过复杂，规则太过严格，以至于任何与样本数据稍有不同，就判断不属于此类，例如决策树过深、回归参数过多等；
二是训练集的问题，或者训练集太小，或者训练集不具有代表性，或者训练集存在噪音但为了拟合异常点而偏离了正常分布。

综上所述，决策树的优点缺点，便能很清楚地明白了。

决策树的优点：训练时间较短、复杂程度较低、预测过程较快、模型容易展示（用树枝状的可视化呈现）等。

决策树的缺点：容易发生过度拟合，解决方式之一是剪枝（例如，在 rpart package 中的 prune 函数），但仍然不够稳定。解决方式之二是随机森林。

四、随机森林决策树

随机森林的决策树，是指：用随机方式建立一个森林，森林里面有很多的决策树，每一棵决策树之间没有关联。每当输入一个新的样本（数据）时，就让森林中的每一棵决策树分别进行判断，最后被判断到某类的情况最多，就预测这个样本为那一类。

在前面的讨论中，我们知道过拟合的问题。那么，如果我们此处通过行与列的两种随机采样，保持随机性，那么就能避免过度拟合。

与本节第二部分最近邻的案例相比，我们此次仍然使用 iris 数据集，但是采用随机森林决策树的方式。

```
#加载 randomForest 包。
library(randomForest)

#按照前面步骤，调用 iris 数据集。
nihao=read.csv("d:/mydata/iris.csv")
```

```
str(nihao)
attributes(nihao)
```

随机森林的决策树，有两个重要原理（步奏）：

（一）随机采样过程：行的采样具有“放回”动作，也就是采样所得的样本集合，有可能有重复的样本；输入样本为 N 个则采样样本也为 N 个；在训练时，每棵决策树的输入样本都不是全部样本。列的采样，是从 M 个特征（feature）中，选择 m 个（m << M）。

（二）枝节完全分裂：对采样之后的数据使用完全分裂的方式建立出决策树，使得每颗决策树的每个子节点无法继续分裂或者里面的所有样本都指向同一个类。

所以，我们仍然首先需要拆分样本为测试集与训练集，接着使用随机森林的方式，预测分类结果（因为是有监督学习，所以实际上是有分类结果的正确答案，那么之后可以对照之后，知晓预测准不准确）。

```
#将 iris 数据集，分为训练集和测试集。
id <- sample (2, nrow(nihao), replace = TRUE,
    prob =c(0.65, 0.35))
trainingdataset <- nihao[id ==1,]
testingdataset <- nihao[id ==2,]

#使用数据集中的其他所有变量，预测 iris.setosa 值。
RADOMFOREST <- randomForest(Iris.setosa~ ., data=trainingdataset, ntree=100,
proximity=TRUE)
## 在 iris.setosa~之后的一点，代表所有其他变量。
## ntree 意指：决策树的数量，要种多少棵树。
## proximity 表示邻近矩阵（proximity matrix），即：任意两个观测实例之间的相似度，
如果两个观测实例落在同一棵树的同一个叶子节点的次数越多，则这两个实例的相似度越
高。除以 ntree 就会把矩阵进行标准化。
```

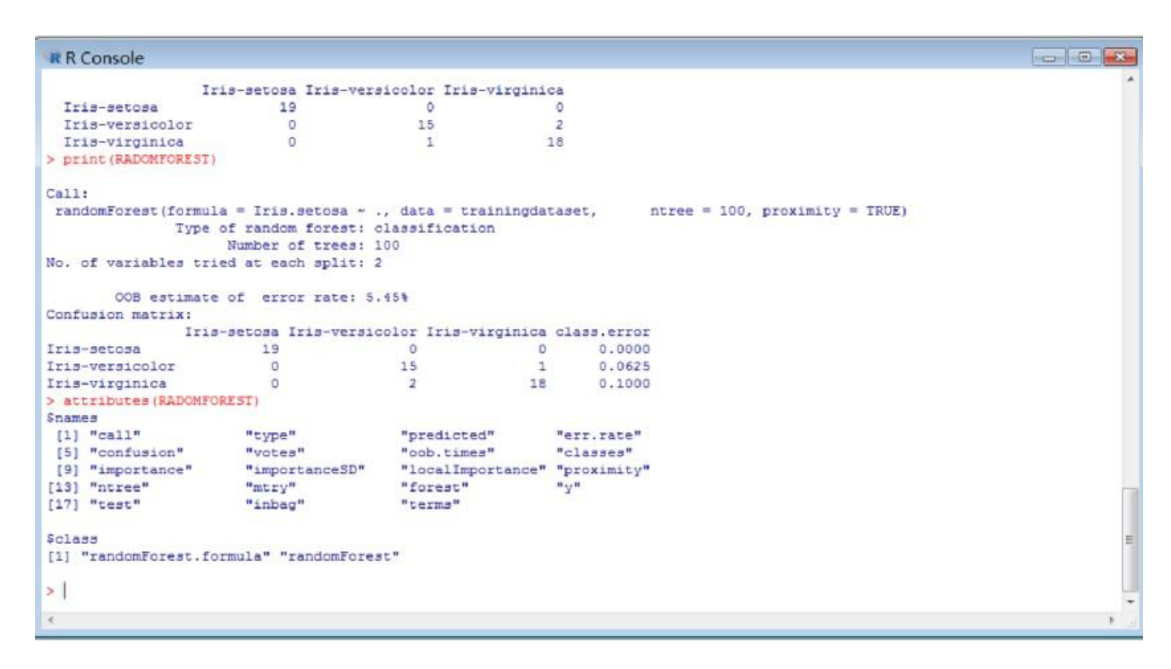
接下来，我们命令 R 输出随机森林 RADOMFOREST 的预测情况、决策树规则、属性。

```
#预测情况的比较
table(predict(RADOMFOREST), trainingdataset$Iris.setosa)

#决策树的规则
print(RADOMFOREST)

#决策树的属性
attributes(RADOMFOREST)
```

结果：



```
                  Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              19               0              0
  Iris-versicolor           0              15              2
  Iris-virginica            0               1             18
> print(RADOMFOREST)

Call:
 randomForest(formula = Iris.setosa ~ ., data = trainingdataset,      ntree = 100, proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 5.45%
Confusion matrix:
                Iris-setosa Iris-versicolor Iris-virginica class.error
Iris-setosa              19               0              0      0.0000
Iris-versicolor           0              15              1      0.0625
Iris-virginica            0               2             18      0.1000
> attributes(RADOMFOREST)
$names
 [1] "call"            "type"            "predicted"       "err.rate"
 [5] "confusion"       "votes"           "oob.times"       "classes"
 [9] "importance"      "importanceSD"    "localImportance" "proximity"
[13] "ntree"           "mtry"            "forest"          "y"
[17] "test"            "inbag"           "terms"

$class
[1] "randomForest.formula" "randomForest"

>
```

图 6.57 随机森林决策树（1）：决策树的预测情况、规则和属性

图 6.57 里的内容，在下方有两条信息 randomForest.formula 和 randomForest 意指：RADOMFOREST 是描述模型被拟合的公式，来自 randomForest 程序。这就是我们刚刚要求 R 显示的属性（Attributes）。

接下来，我们要求 R 输出误差率（随机森林决策树到底预测的准不准）以及变量重要性（我们研究的目的就是为了找到众多变量当中，能够帮助我们进行决策的变量）的图表。

```
#预测情况的比较（表格，在图 6.57 所以这里是：散点图）
table(predict(RADOMFOREST), trainingdataset$Iris.setosa)

#决策树的规则
print(RADOMFOREST)

#决策树的属性
attributes(RADOMFOREST)
```

结果：

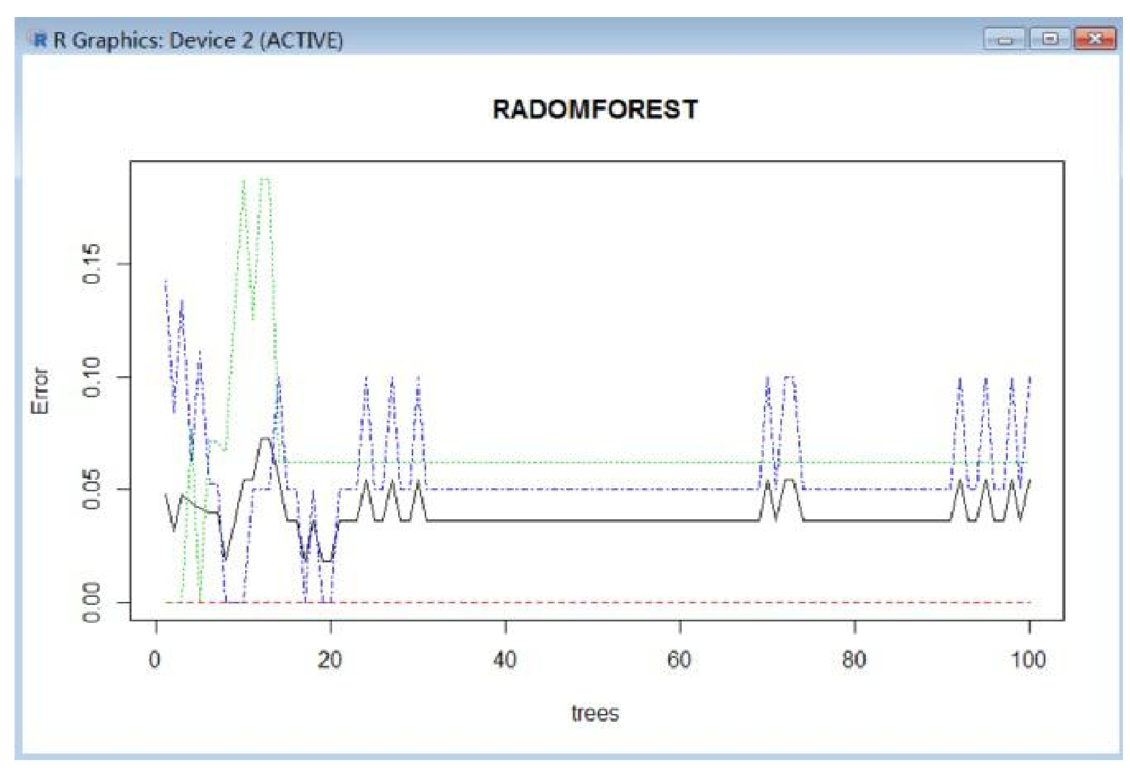


图 6.58 随机森林决策树（2）：随机森林的不同决策树的误差率

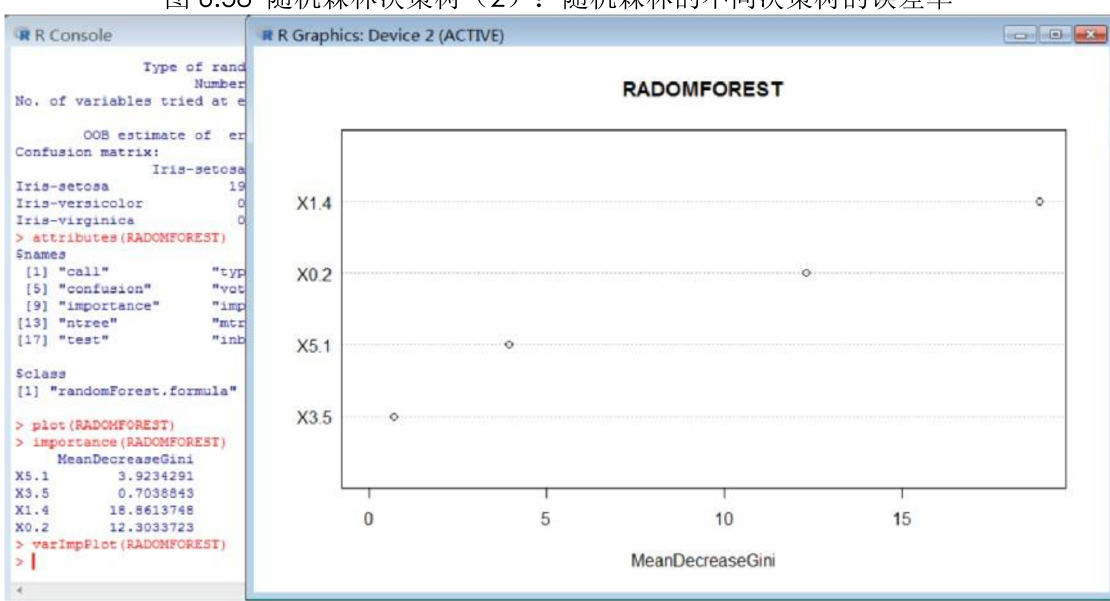


图 6.59 随机森林决策树（3）：变量重要性的表格与散点图

接下来，最后一个步奏，即：测试决策树！代码如下：。

```
# 使用测试集，对已经建好的随机森林的决策树，进行测试。
iris_test <- predict(RADOMFOREST, newdata = testingdataset)
table(iris_test, testingdataset$Iris.setosa)

# 通过绘图 plot 的函数 margin 查看结果。
plot(margin(RADOMFOREST), testingdataset$Iris.setosa)
## margin 边距
```

结果：

```
> iris_test <- predict(RADOMFOREST, newdata = testingdataset)
> table(iris_test, testingdataset$Iris.setosa)

iris_test         Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              18               0              0
  Iris-versicolor           0              19              2
  Iris-virginica            0               3             19
```

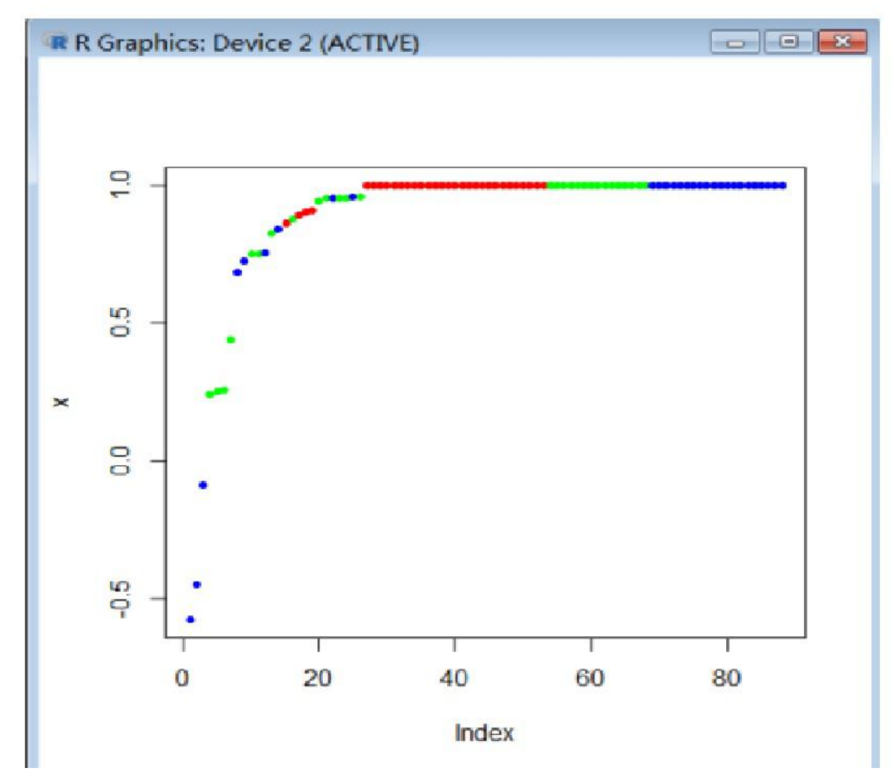


图 6.60 随机森林决策树（4）：测试决策树

我们知道：数据点的边距为正确归类的比例，减去被归类到其他类别的最大比例；边距如果是正数，一般而言，数据点的归类可以判断是正确。那么，根据图 6.60 所示，决策树的预测效果还是非常准确的。

我们使用 randomForest 包的函数 randomForest()存在两个限制：一是无法处理带有"缺失值（Missing Data）"的数据集，因此要先做缺失值的数据推定；二是分类属性的层级数量，以 32 为最大值，因此超过 32 级别的属性首先需要进行转换才能使用该函数。目前我们使用 R 作为随机森林决策树的时候，在操作技巧上，需要注意两件事：

（1）使用 party 包的函数 cforest()能够建立随机森林，而且没有限定分类属性的层级数量，不过，多级分类属性需要更多内存空间，建立随机森林的时间更长。

（2）使用 party 包的函数 ctree()目前还不能够很好地处理缺失值，含有缺失值的实例，根据替代规则，有时在左，有时在右。

另外，我们使用 party 包的函数 ctree()构建决策树时，如果训练集的某个变量在建立决策树之后被剔除了，在使用测试集进行预测时，仍然需要包括该变量，否则调用函数 predict()会失败。

如果测试集与训练集的分类变量级数不同，测试集的预测也会失败。解决方案是：使用训练集建立一颗决策树后，再使用第一颗决策树所包括的所有变量，重新调用 ctree()建立

一颗新的决策树，在新的决策树的建立过程中，根据上一步的测试集的分类变量级数，来设置训练数据集。

本章小结

本章介绍机器学习，从历史发展和模式识别的角度，介绍机器学习作为数据集的特征抽取的含义，接着通过算法选择以及项目实践，说明交叉验证（上接第六章）的方式。在第二部分主要介绍无监督学习，以及第三部分介绍有监督学习（接着第九章）。本章暂且以 R 为主，以及给于一些练习用途的小数据集，这样，便于操作和练习，主要目的在于理解原理、熟悉过程，以及掌握一些经常使用到的机器学习方法。本章讲述了以下内容：

（一）机器学习的历史、任务、过程等，主要概念是：在经典统计（第四章）之外，如果需要进行科学性的数据分析，那么主要的问题是模式识别，模式识别的主要难点是特征抽取，而这就是机器学习要做的事情。

（二）无监督学习和有监督学习的差别，主要是：机器学习需要区分训练集和测试集，训练集用来训练模型得到参数，有了参数就能完成数据建模（第六章）的任务，完成任务之后，需要使用测试集进行测试，看看是否预测准确。在有监督学习的数据集里，都有标准答案，也就是说，训练集可以自己先对答案，自己先做一番调整，再给测试集进行测试；然而，再无监督学习的数据集里，没有标签，所以，训练集完成模型之后，直接给测试集测试，人们只要“觉得”结果没有差距太大就算完成检验了。

（三）主成成分分析：经过降维之后，找到能够足以代表或者覆盖整个数据集的关键少数，即，作为主要成分的变量及其数值。我们介绍了 K-mean 和 PCA 两种常见的主成成分分析的方法。

（四）因子分析：经过降维之后，通过旋转因子矩阵轴，相当于从各种不同的维度角度重新审视和定位坐标，给于原来数据集中对应在空间中的每个点的各自新的坐标下的位置，使得最大程度上，能够以新的坐标找到能够覆盖全数据集的少数关键。

（五）因子分析与主成分分析的综合应用：先通过旋转因子矩阵轴，找到关键少数，再把关键少数进行建模，使得所创造出来的新的变量，能够解释绝大多数的数据集的各个数值变化的情况。

（六）关联分析：根据变量间的相关性，把两组变量（不是两个变量）的关系集中到少数综合变量上，求得能够解释数据集的参数以及模型。

（七）贝叶斯：朴素贝叶斯通过对于过去已经发生时间的数据来推断未来事件发生的概率。根据贝叶斯方法建立的分类器，它利用训练集以及特征取值，来计算每个类别被观察到的概率。

（八）最近邻：最近邻 kNN 算法将对象视为一个多维特征空间（feature space）内的坐标上的一个点，计算距离函数（distance function）或者两者之间相似性。

（九）决策树：在众多变量当中，找到【决策】所需要考虑的【变量】的【优先次序】和【等级次序】，通过剪枝【去过拟合】（去除掉数据过度模拟符合模型的情况）。

（十）随机森林决策树：用随机方式建立一个森林，森林里面有很多的决策树，每一棵决策树之间没有关联。通过检验找到最适合的决策树。

在此前章节的练习中，我们已经具备基础阅读 R 与 Python 代码的能力，因此本章直接使用 R 进行示例，如果使用 Python 则有可能增加操作上的复杂度。下次见到 R 会在第十章。

习　　题

一、单选题

1、有监督学习【不包括】以下哪个方法（___）

（1）朴素贝叶斯

（2）随机森林决策树

（3）最近邻

（4）因子分析

2、无监督学习【不包括】以下哪个方法（___）

（1）主成成分分析

（2）关联分析

（3）随机森林决策树

（4）因子分析

二、多选题

1、机器学习项目的实施顺序是（___）

（1）准备数据

（2）定义问题

（3）评估算法

（4）改良结果

2、机器学习模型的训练顺序是（___）

（1）探索数据

（2）训练模型

（3）评估模型

（4）提高性能

三、问答题

1、 请说明有监督学习和无监督学习的不同？

2、 请介绍一个有监督学习的算法案例。

3、 请介绍一个无监督学习的算法案例。

四、课后作业

基于十篇科学论文的学习之后，请按照 7.1.4. 项目练习的最终呈现结果的内容，撰写一个科学论文的研究设计的架构，以.txt 或者.doc 格式的文件写下，即可。内容包括：

Step 1: 背景脉络

Step 2: 关键问题

Step 3: 解决方案

Step 4: 发现

Step 5: 限制

Step 6: 结论

五、作业释疑

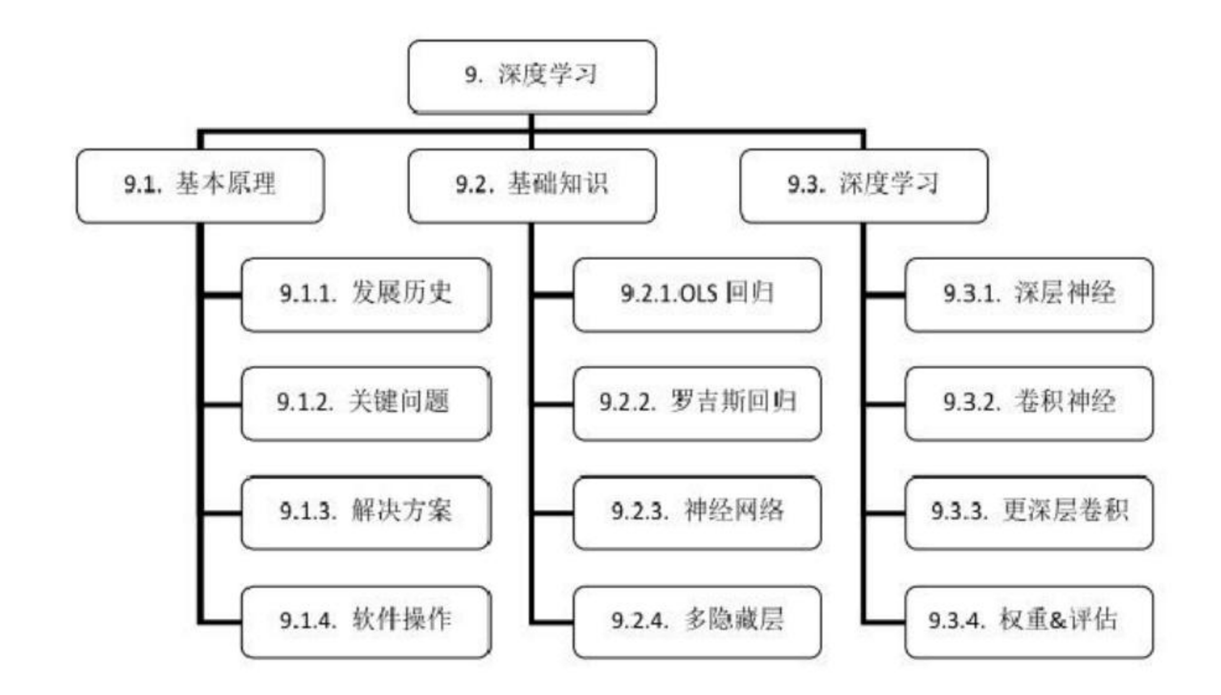
如果没有实际的数据科学的研究，不能写成一篇短篇论文，并不影响作业，因为这次作业的要求是：研究设计。目的是为了结合数据和机器学习的内容，仔细想过一遍。

第七章　深度学习篇

学习目标

理解深度学习所要解决的问题和局限
掌握 Tensorflow 的基本操作。
理解罗吉斯回归、神经神经和深层神经网络。
掌握 TF 进行 OLS、神经神经和深层神经网络的操作。
理解卷积神经网络以及权重和评估方式。
掌握 TF 进行完整的深度学习的操作过程。

知识结构




导入案例

本章节开始，我们使用 Python 加载 TensorFlow2.0 的库，从基本概念开始有一理解之后，接着进行 TensorFlow 的基本操作，在第二节的部分，回顾了第四、六、八章？的部分内容，把概念借由 TF 操作重新组织串联起来之后，在第三节的部分进行深度学习的实际操作，从做中学的方式，予以理解消化。

第一节 基本原理

一、发展历史

深度学习是基于机器学习延伸出来的一个新的领域，由以人大脑结构为启发的神经网络算法为起源加之模型结构深度的增加发展，并伴随大数据和计算能力的提高而产生的一系列新的算法。著名科学家 Geoffrey Hinton 等人在 2006 年和 2007 年在《Sciences》等上发表的文章被提出之后逐步兴起。

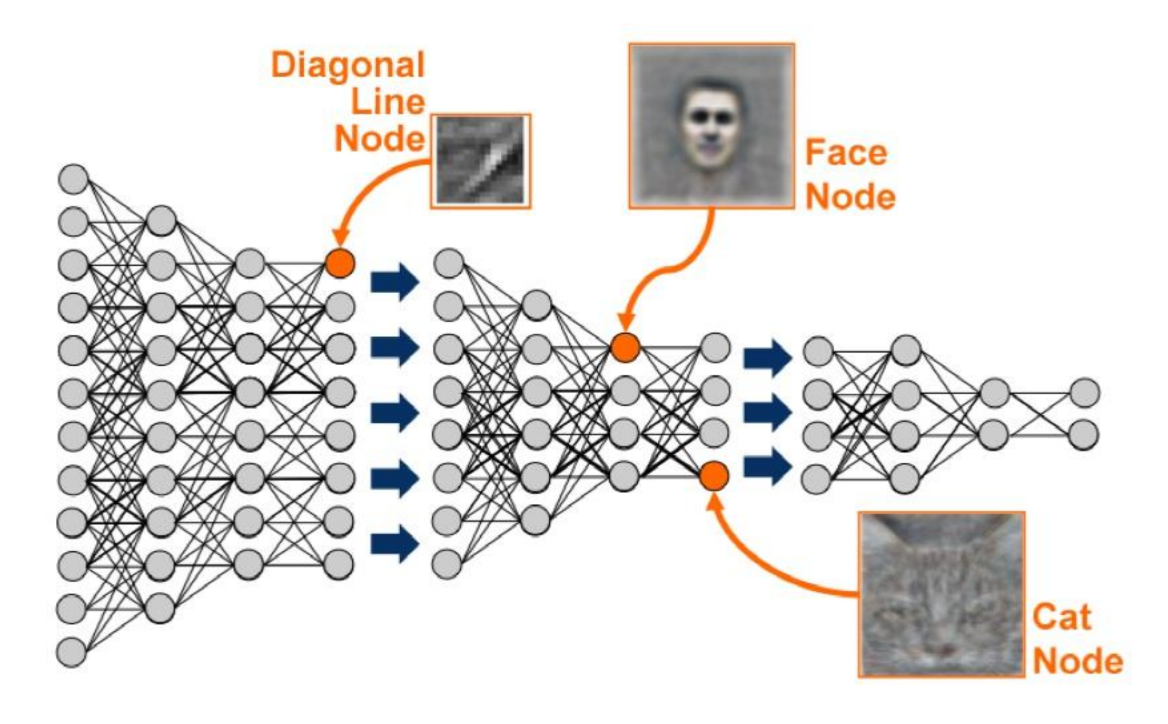


图 7.1 深度学习示意图

深度学习是机器学习中延伸出来的一个领域，被应用在图像处理与计算机视觉、自然语言处理以及语音识别等领域。自 2006 年至今，学术界和工业界合作在深度学习方面不断取得了突破性进展。学校以斯坦福大学、多伦多大学、纽约大学为代表，工业界以 Google、Facebook 和百度为代表走在深度学习研究与应用的前沿。

目前我们使用的华为手机的语音识别、小米手机相片人脸识别、金融机构和安保机构的人脸识别、百度的自动汽车驾驶和无人飞机、搜狐的图片搜索等都已经使用到了深度学习技术。在可预见的未来，无人驾驶汽车中的路标识别、图片识别、语音识别、语言生成文字等当前生活中的应用，其影响范围和对于人类社会的影响深度，也会越来越广和越来越多。



图 7.2 深度学习应用示意图

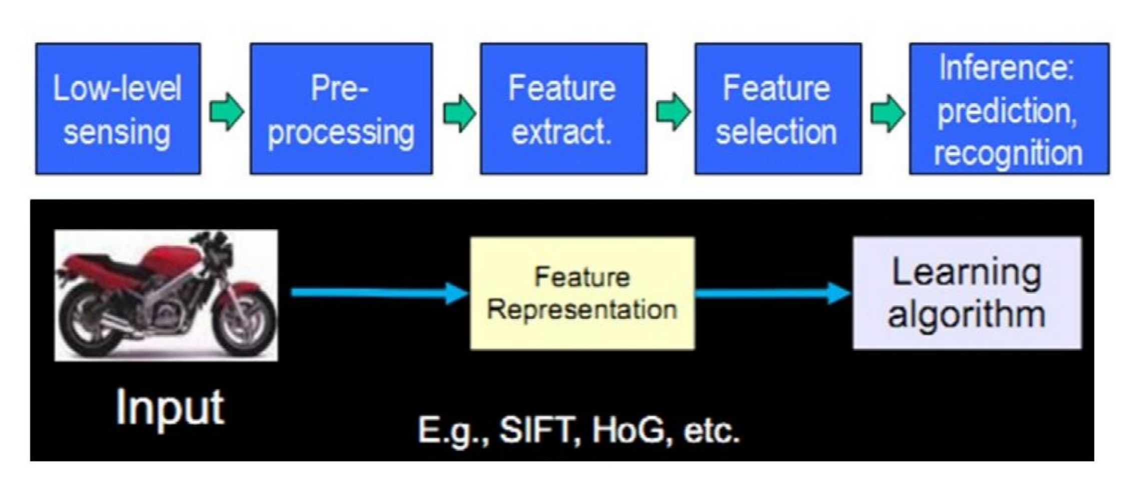


图 7.3 模式识别的应用



图 7.4 商业应用的需求

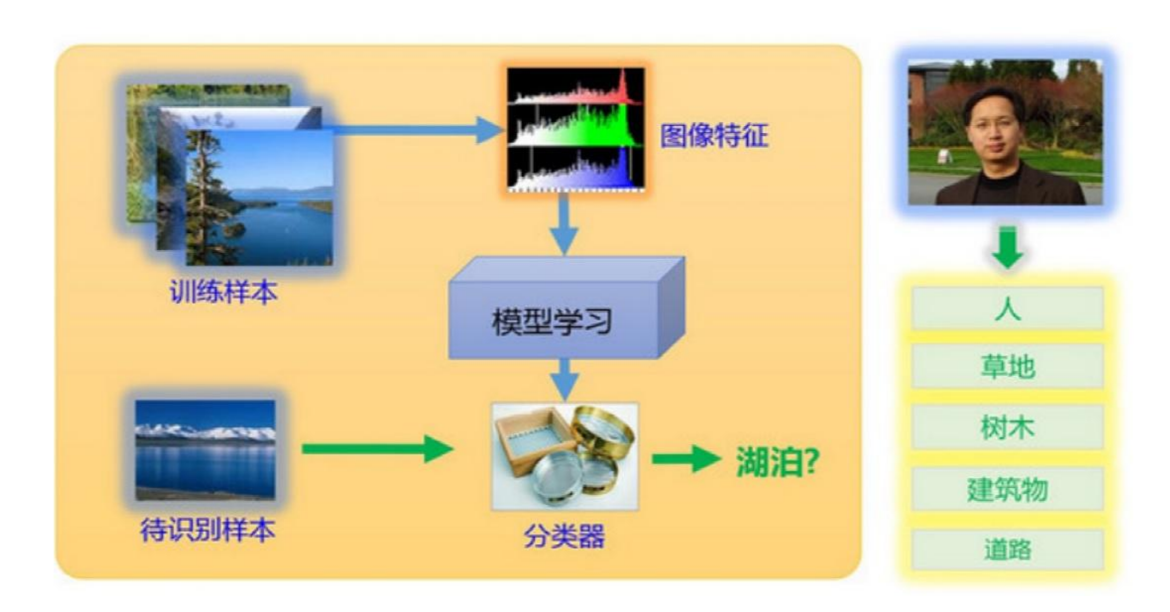


图 7.5 机器学习的进步

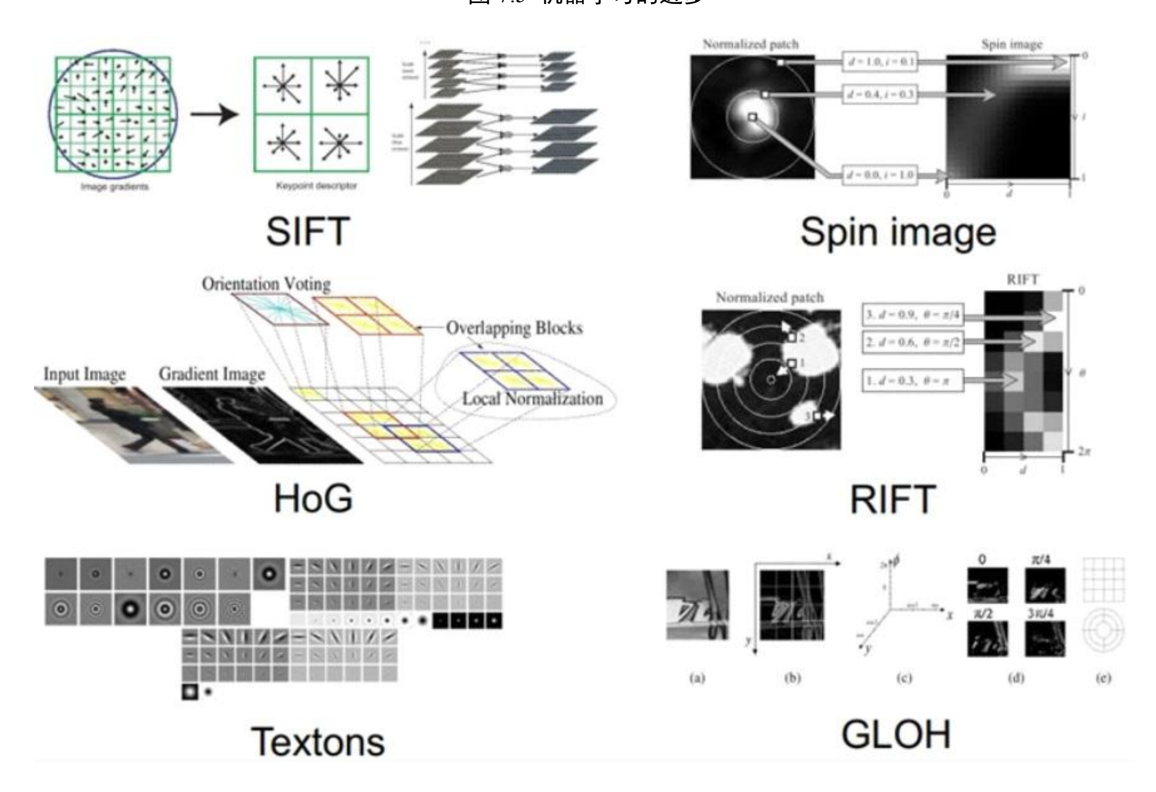


图 7.6 模式识别的进步

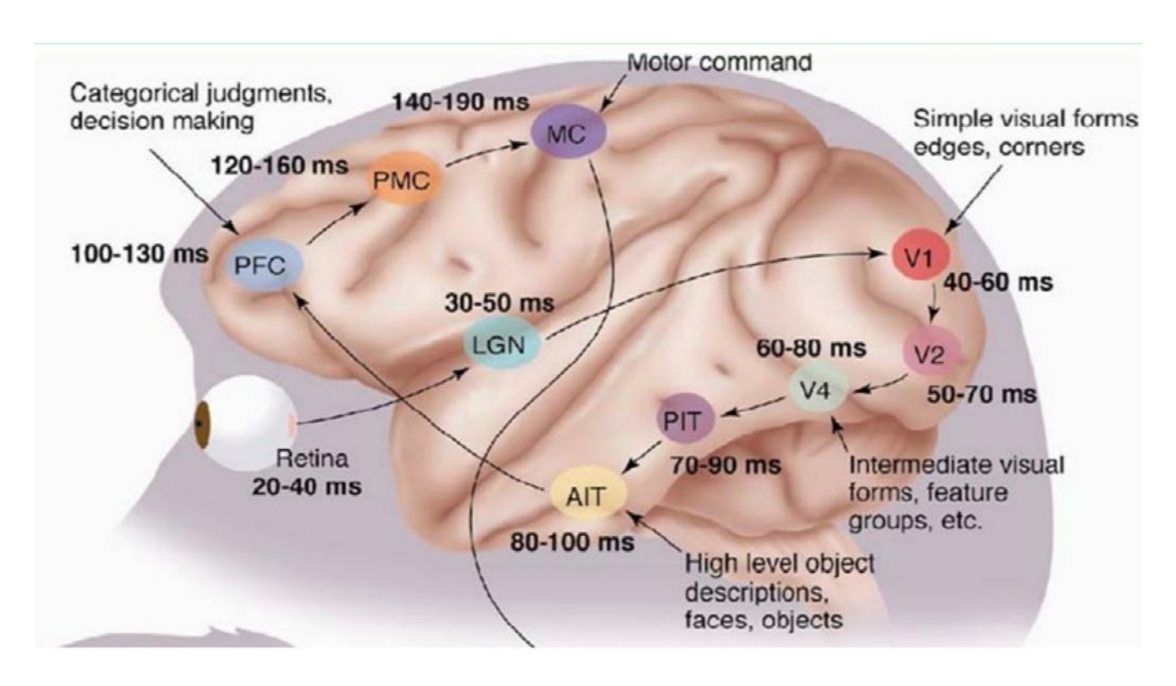


图 7.7 脑科学研究的成就

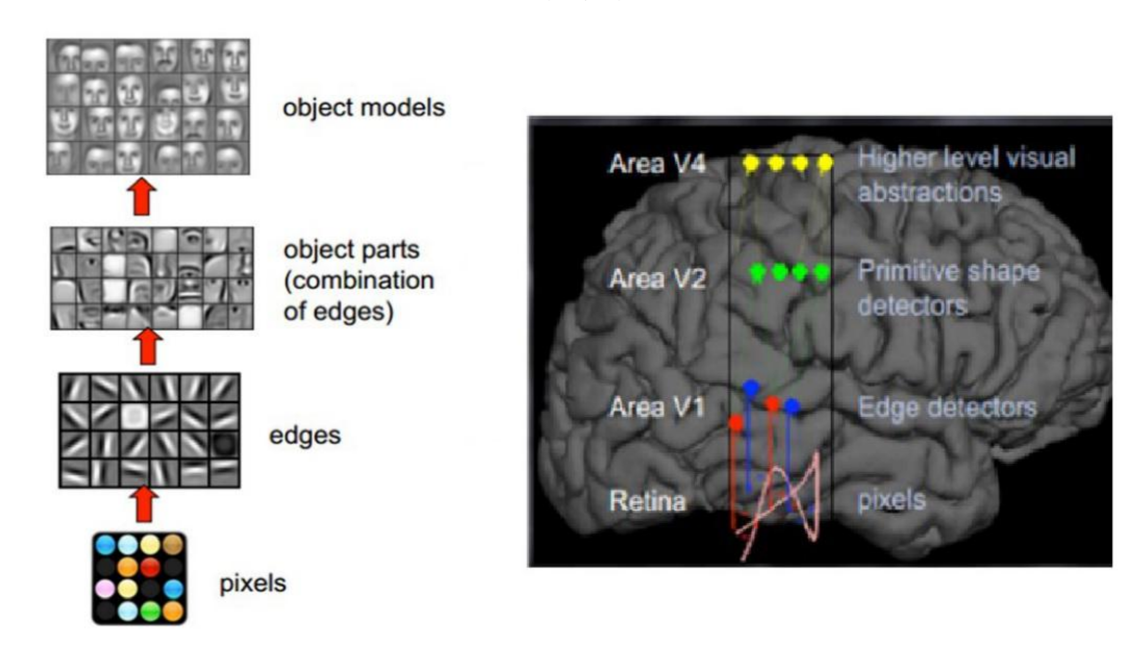


图 7.8 脑科学与模式识别的结合

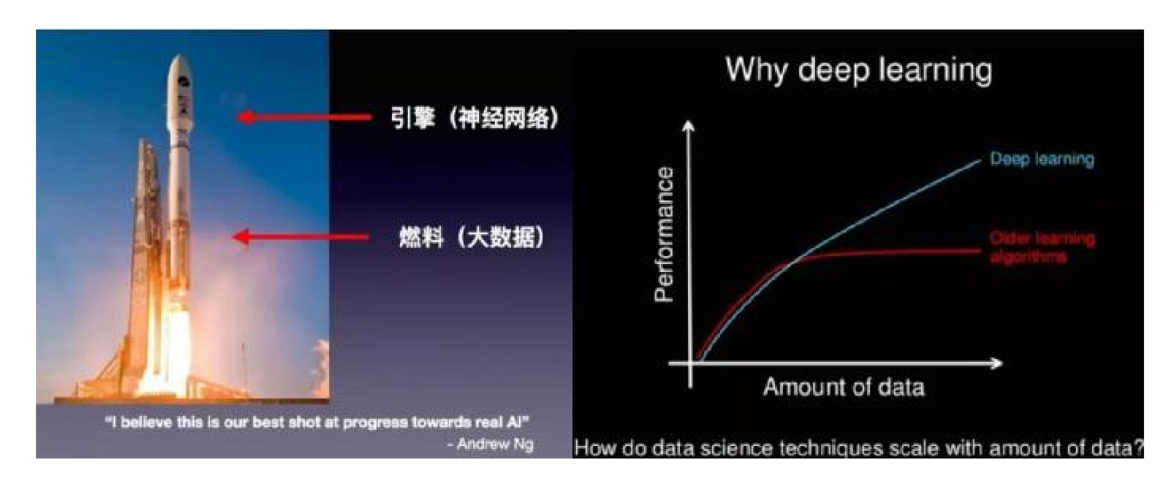


图 7.9 计算能力提高与普及

神经网络硬件加速器

- CPU
- DSP(数字信号处理器)
- GPGPU(通用图形处理器)
- FPGA (现场可编程门阵列)

项目	型号	价格
CPU	Intel i7-6850K	¥4,699
主板	华硕（ASUS）X99-DELUXE II 主板	¥3,995
机箱	乔思伯（JONSBO）W2 黑色 全塔式机箱	¥519
电源	美商海盗船（USCorsair）额定650W RM650x 电源	¥869
内存	金士顿(Kingston)骇客神条 Fury系列 DDR4 2400 64G (16GBx4)	¥2,459
散热器	猫头鹰（NOCTUA）NH-D9L CPU散热器	¥484
显卡	EVGA GTX 1070 8G SC GAMING ACX 3.0 Black Edition	¥3,199
SSD	闪迪(SanDisk) 加强版 480G 固态硬盘	¥680
合计		¥16,904

图 7.10 图论与硬件结合

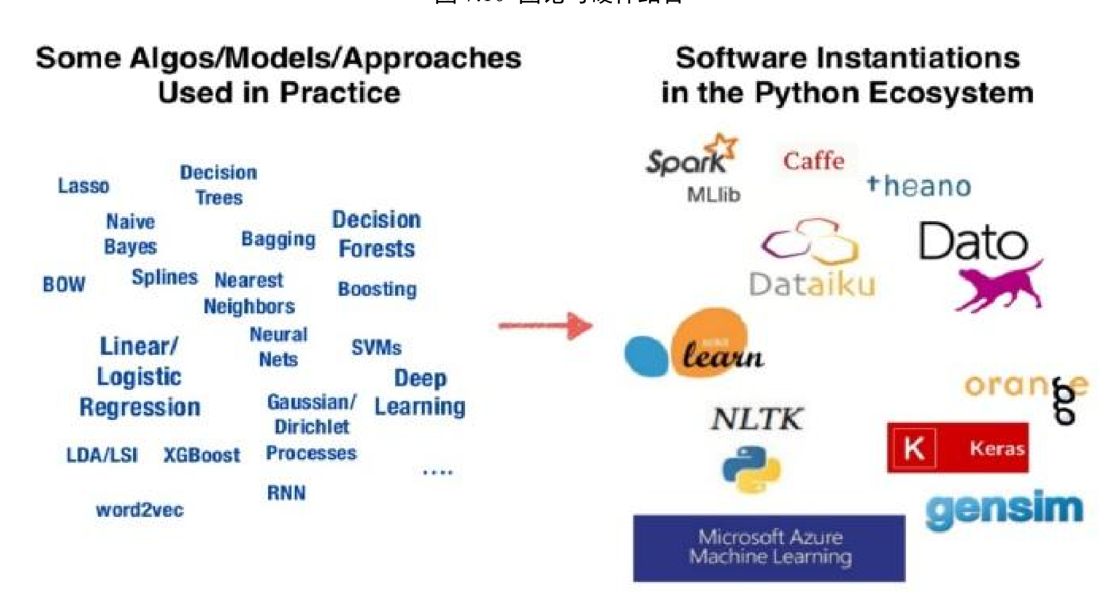


图 7.11 开源软件的兴起

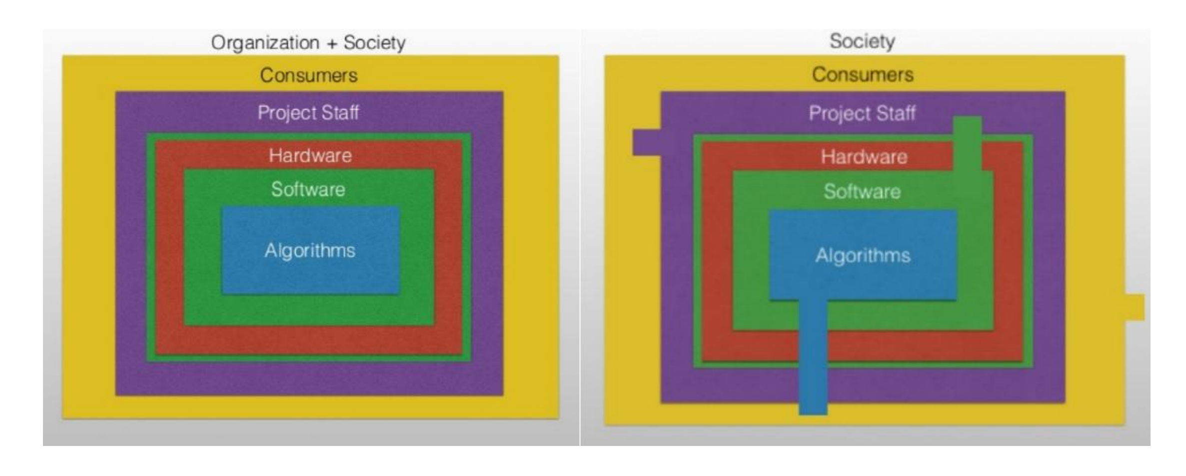


图 7.12 程序就是管理的思想

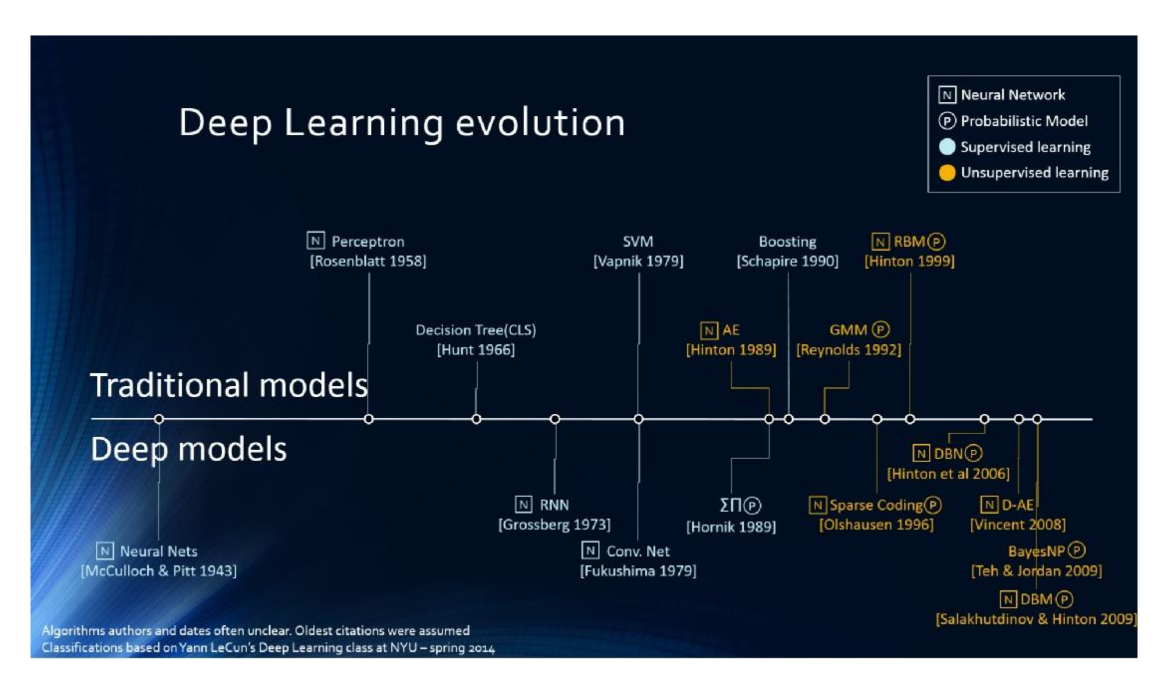


图 7.13 无监督学习的兴起

二、关键问题

深度神经网络主要解决模式识别的问题，那么，具体的关键技术，可以拆解为五个部分，前四个部分是工程实现问题，最后一个步奏是科学辩证问题。

（一）特征向量空间；

（二）浅层特征表示；

（三）特征抽取；

（四）结构化特征；

（五）准确性与复杂性的两难。

如果您考虑前面第一部分机器学习的原理，那么其实上述（一）到（三）的步奏就是一种从特征空间到解题空间的过程，第（四）步骤也是过程之一，但是所不同的是，在机器学习的章节里，我们通过有监督学习进行几次迭代计算之后，便可求得结果，而深度学习由于主要处理无监督学习算法，没有正确的答案的情况，因此需要更多次数的迭代，特别是通过神经网络的方式（在机器学习当中是有监督学习，但是在无监督情况下则应该采用多层神经网络甚至深度学习），因此第（四）在机器学习章节里算是完成，但是在深度学习的章节里，还需要考虑第五步。换言之，因为无法解决第五步，所以才有接下来的神经网络

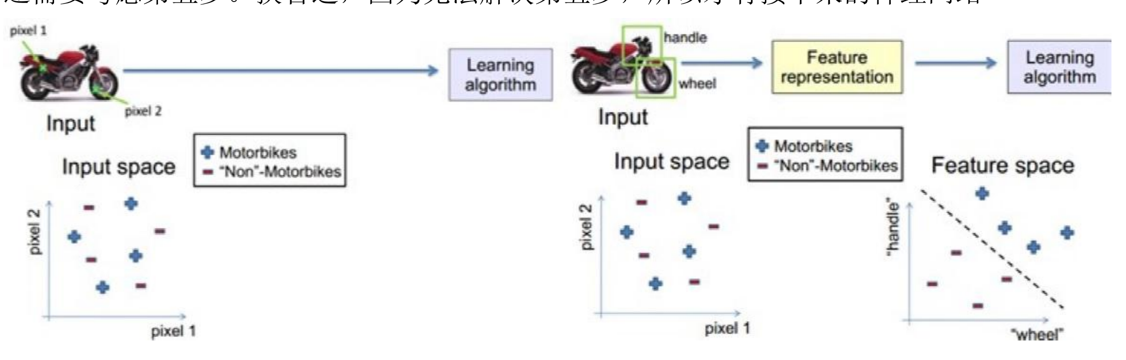


图 7.14 特征向量空间

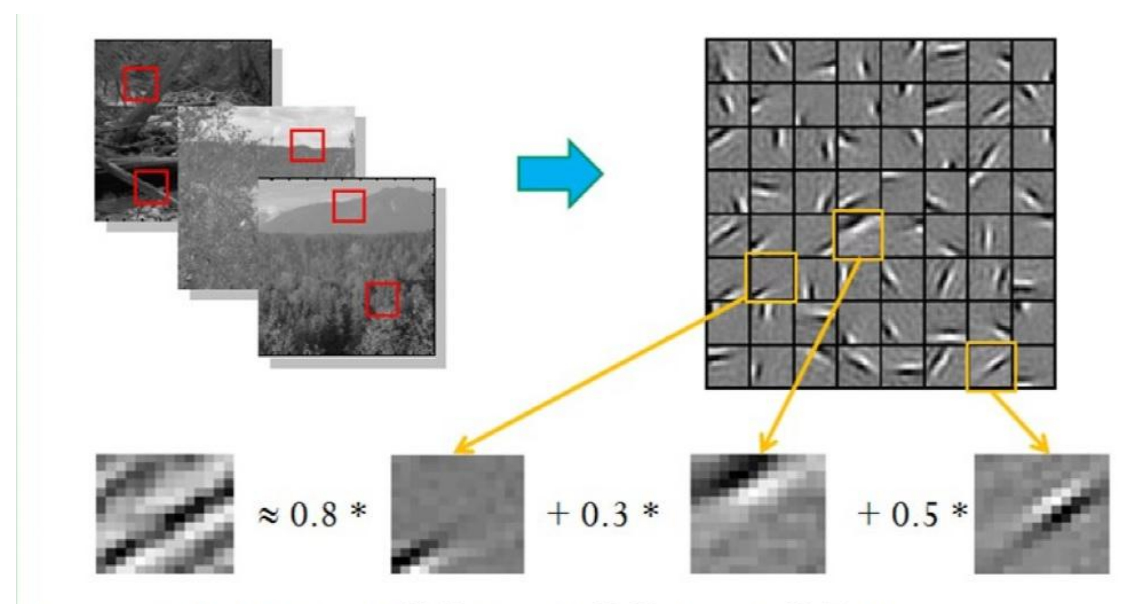


图 7.15 浅层特征表示

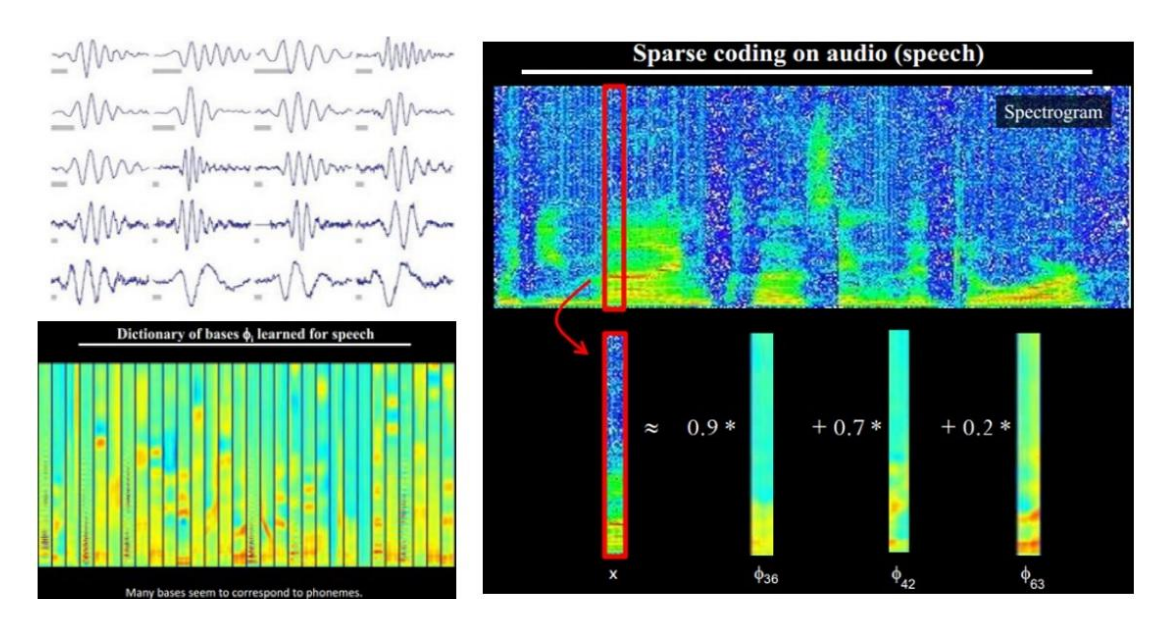


图 7.16 特征抽取

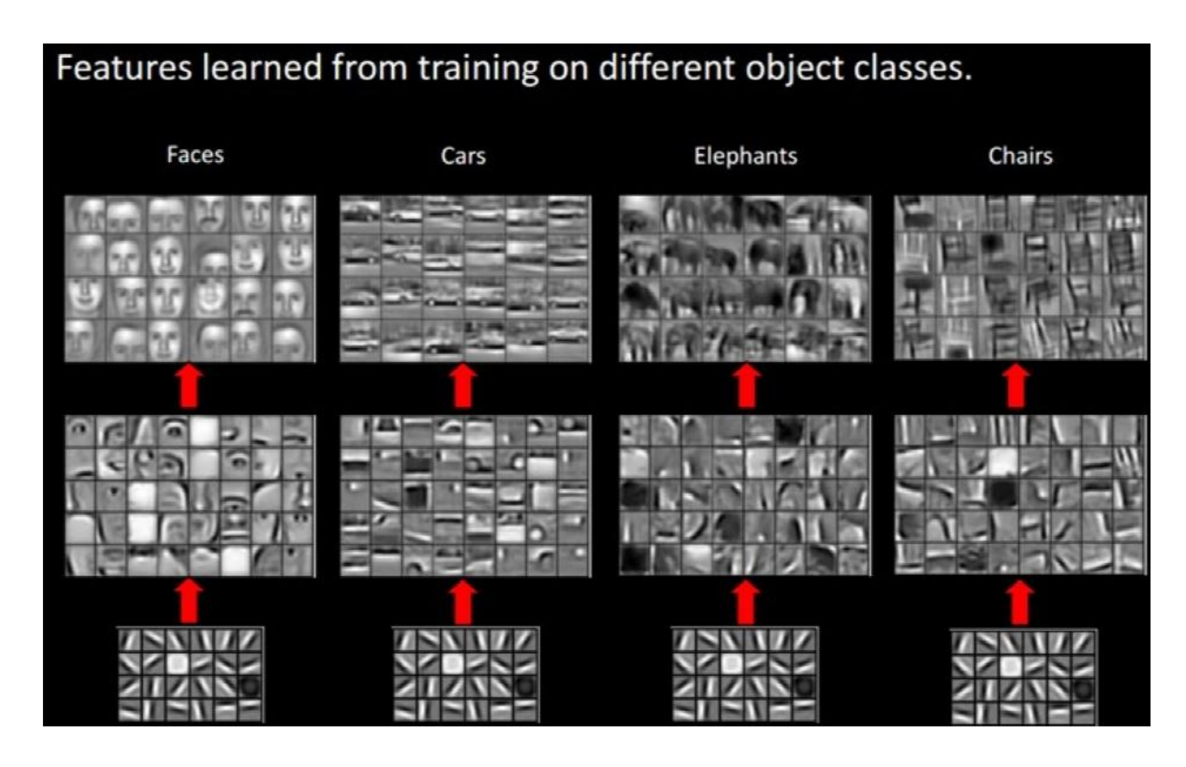


图 7.17 结构化特征

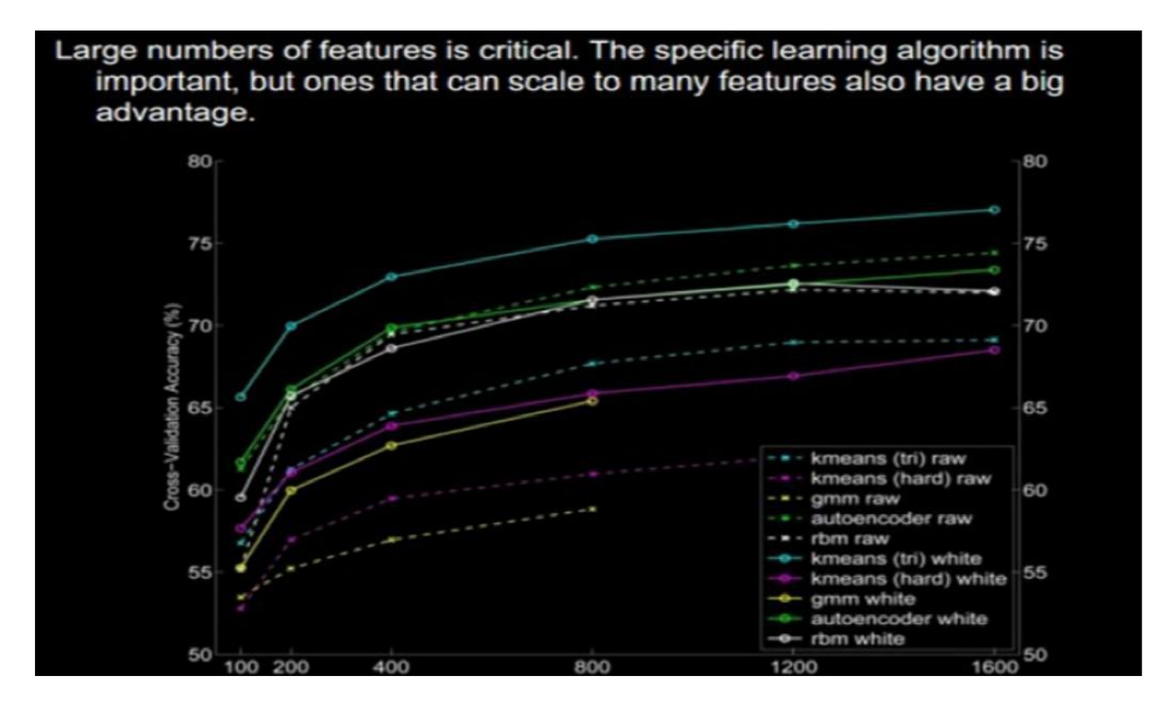


图 7.18 准确性与复杂性的两难

神经网络已在前面的章节中进行过初步的数据建模的角度的介绍，此处我们仍然以它的工作流程，来拆解为：

（一）从逻辑回归到神经元的【感知器】；

（二）浅层神经网络，如能完成求解任务，即可成为数据建模；

（三）深度神经网络，如不能完全求解任务，则考虑进一步这么做；

（四）神经网络适用分类问题，即机器学习当中的分类与回归，此时是做分类，因为无监督以及数据特征的原因；

（五）用 LR 和 SVM 等线性分割的局限，此时仍要考虑其他机器学习算法如 LR 和 SVM 能否解决，或者，不能解决但是可以带给神经网络什么重要的发现和启示意义；
（六）用【神经元】建立【逻辑与】和【逻辑与】，此时的神经网络已经不是和 SVM 等处在同一水平，相互比较谁能解决问题的层面了，而是进一步考虑了如何使用（设计）神经网络开展问题求解的层面了。
（七）用【线性分类器】对平面样本的点分布进行分类，走到这一步，开始考虑究竟是用机器学习足以解决问题，还是必须开始动用深度学习了。
（八）人工智能代理 AI agents 的思路是开始考虑动用深度学习的一个开端。

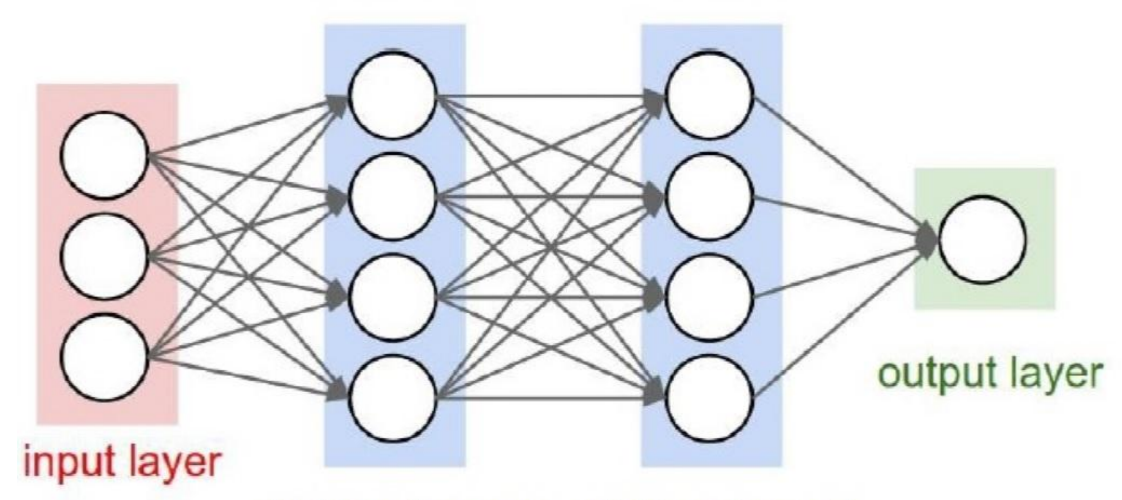


图 7.19 神经网络的图

$$z = \theta_0 + \theta_1 X_1 + \theta_2 X_2$$

$$a = g(z) = \frac{1}{1+e^{-z}}$$

图 7.20 从逻辑回归到神经元【感知器】

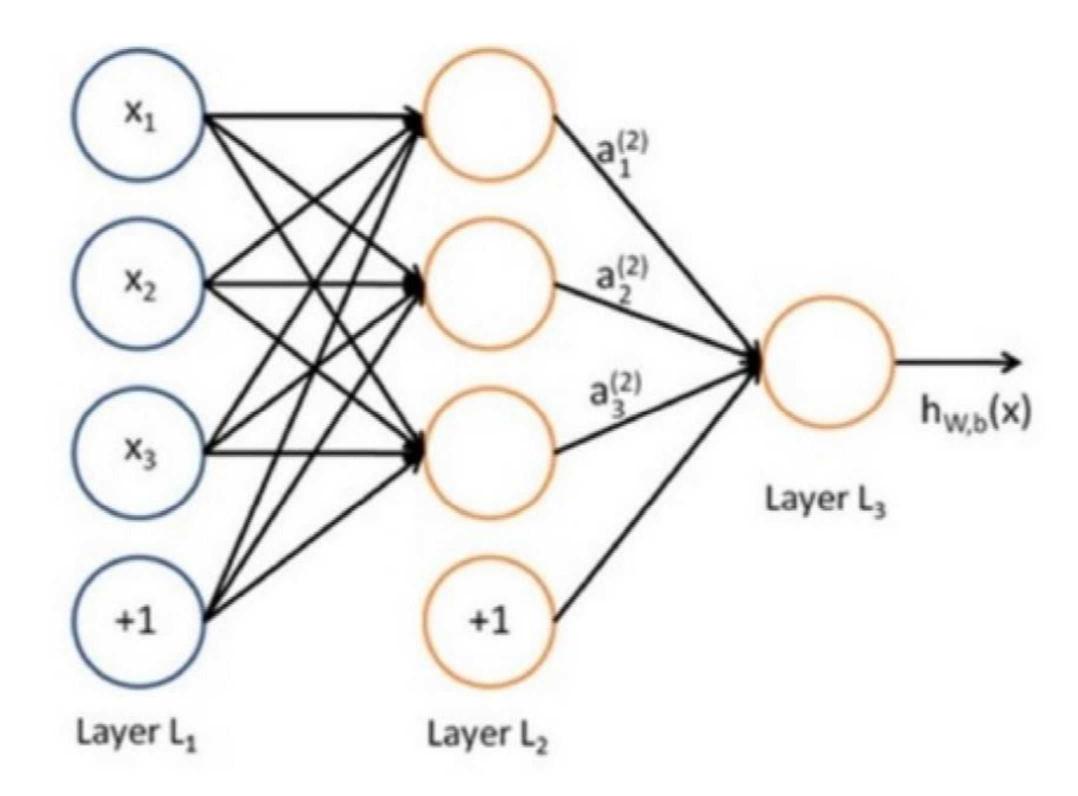


图 7.21 浅层神经网络少量隐层

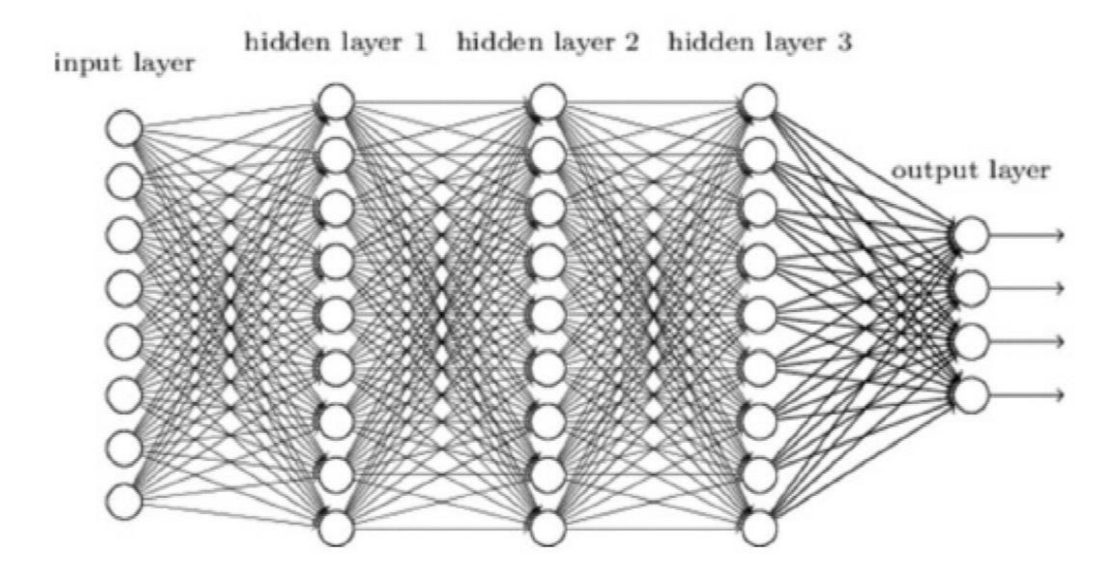


图 7.22 深度神经网络增加隐层

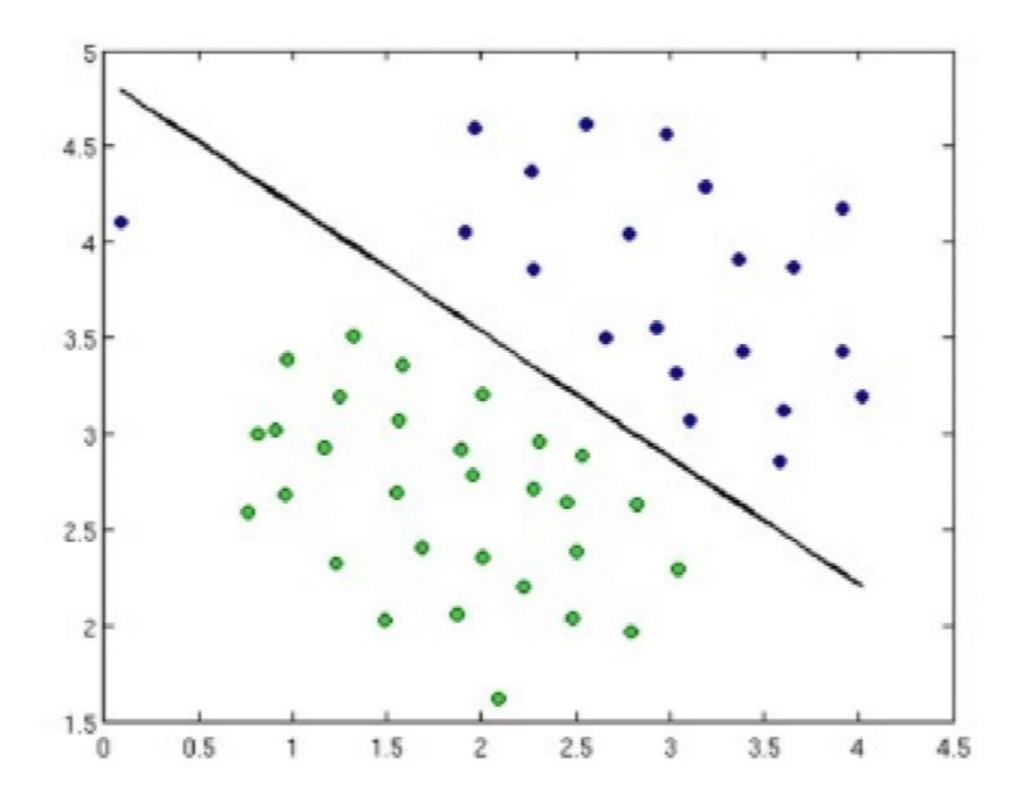


图 7.23 神经网络适用分类问题

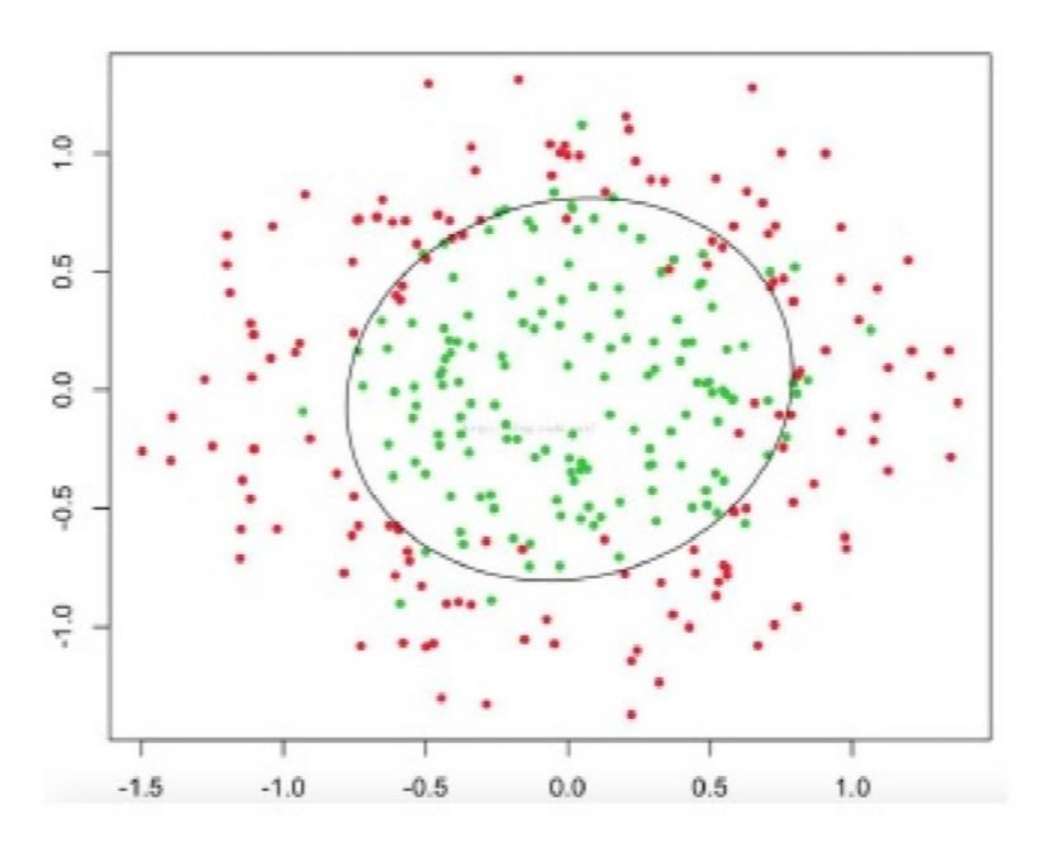


图 7.24 用 LR 和 SVM 等线性分割的局限

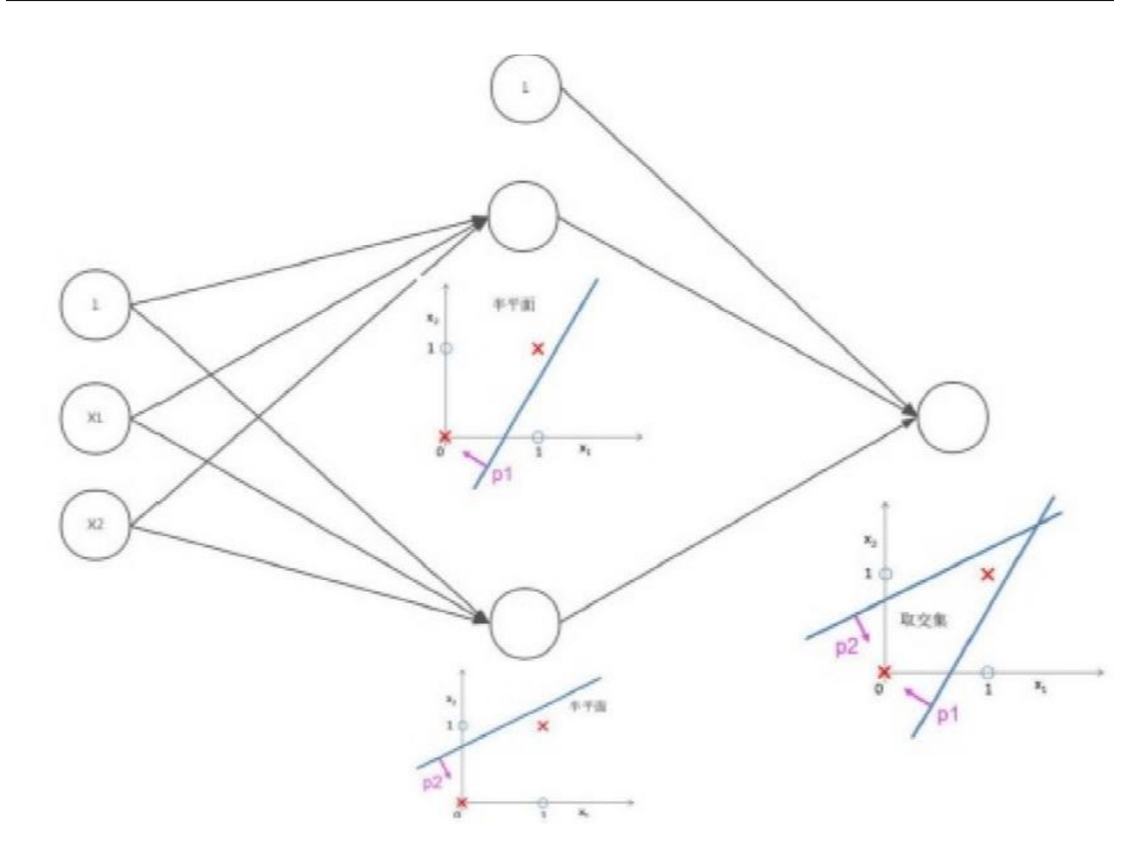


图 7.25 用【神经元】建立【逻辑与】和【逻辑与】

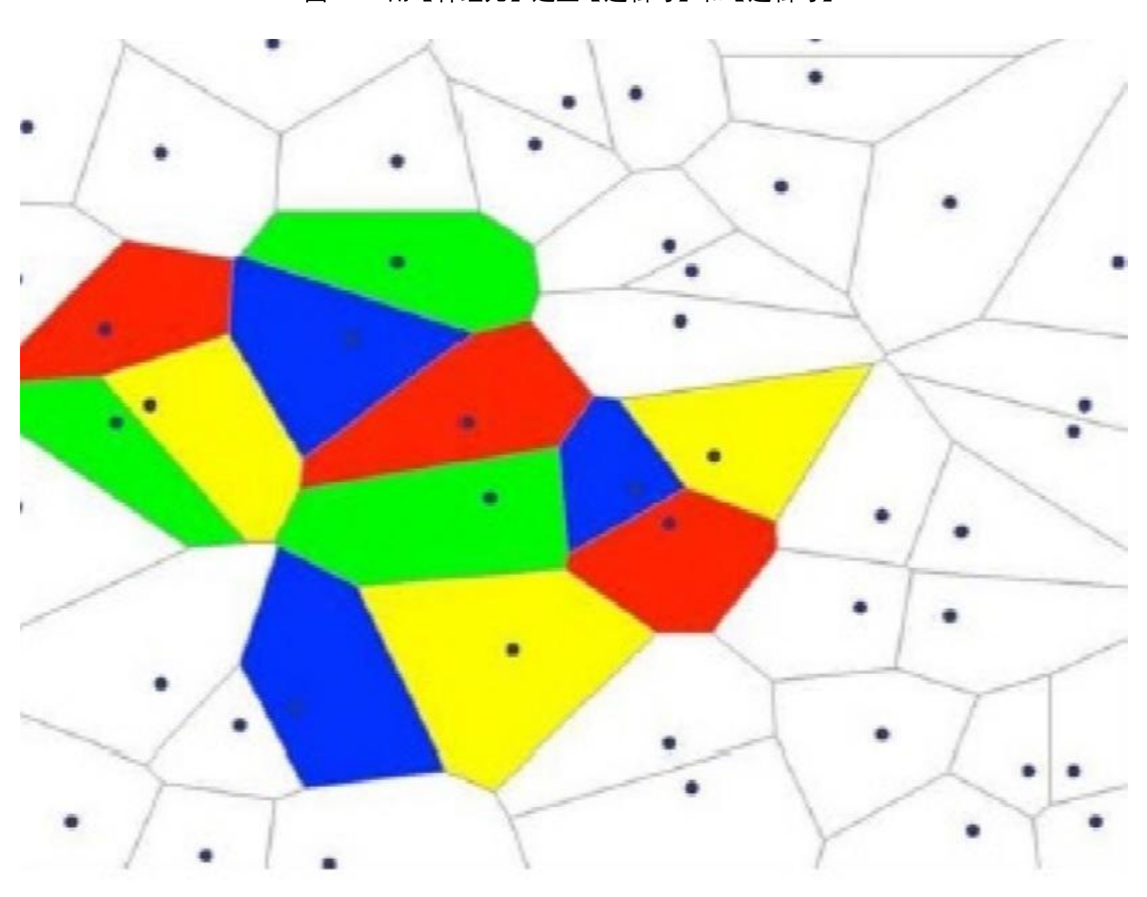


图 7.26 用【线性分类器】对平面样本的点分布进行分类

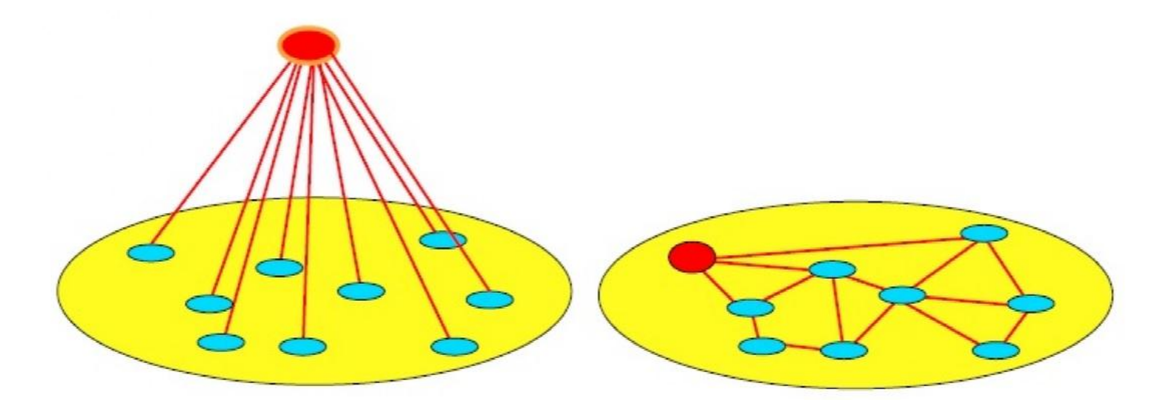


图 7.27 人工智能代理 AI agents 的思路

三、解决方案

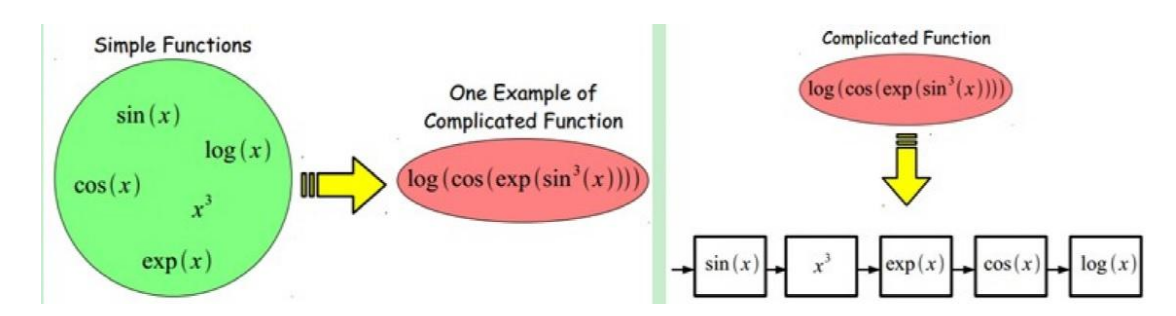


图 7.28 第二次机器学习浪潮

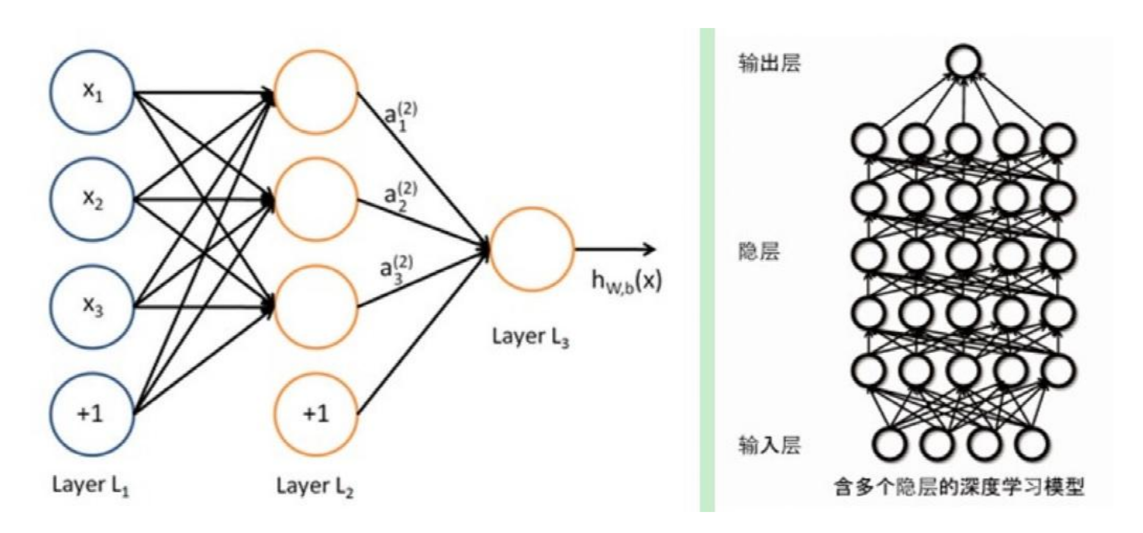


图 7.29 逻辑斯回归的应用

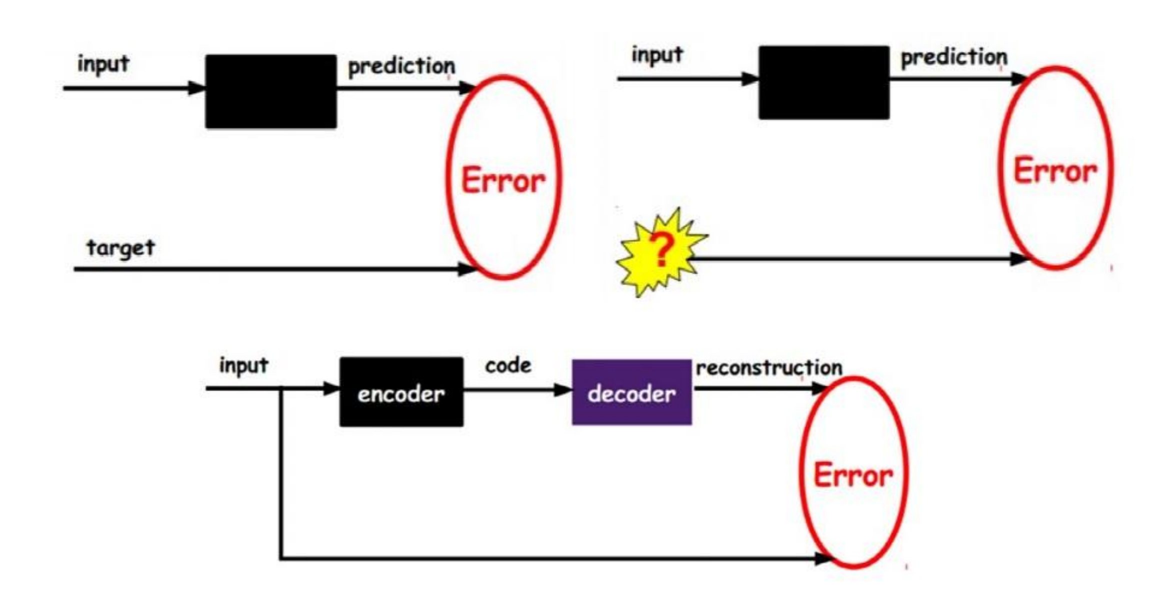


图 7.30 非监督学习（无给定标签）

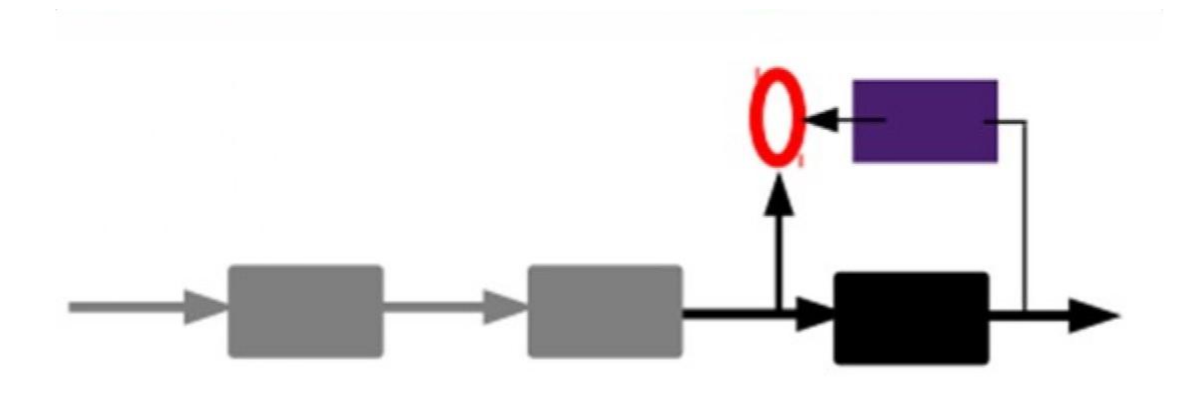


图 7.31 逐层训练的方案

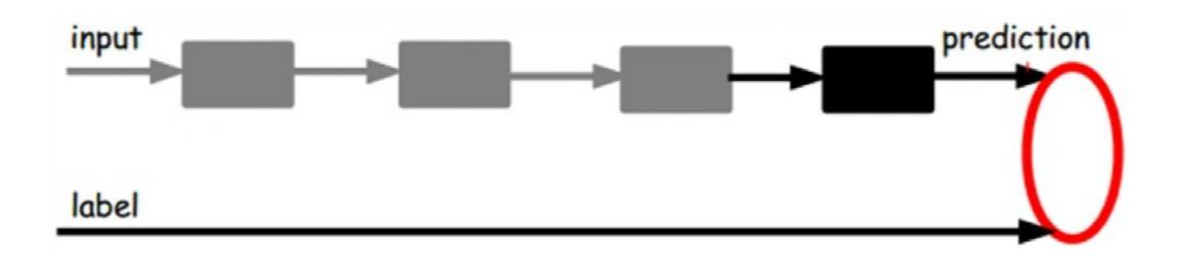


图 7.32 非监督学习（结合监督学习）

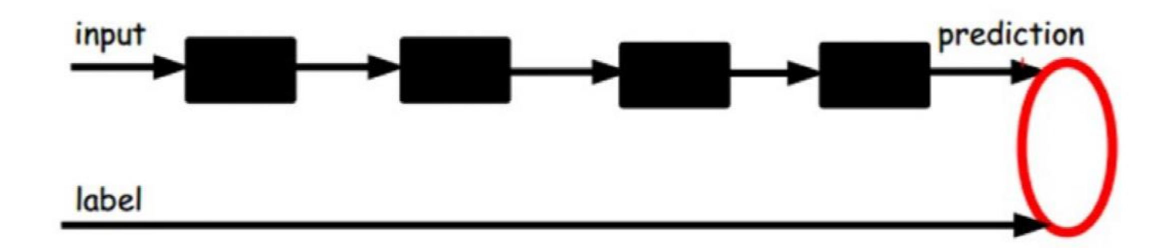


图 7.33 非监督学习（微调系统）

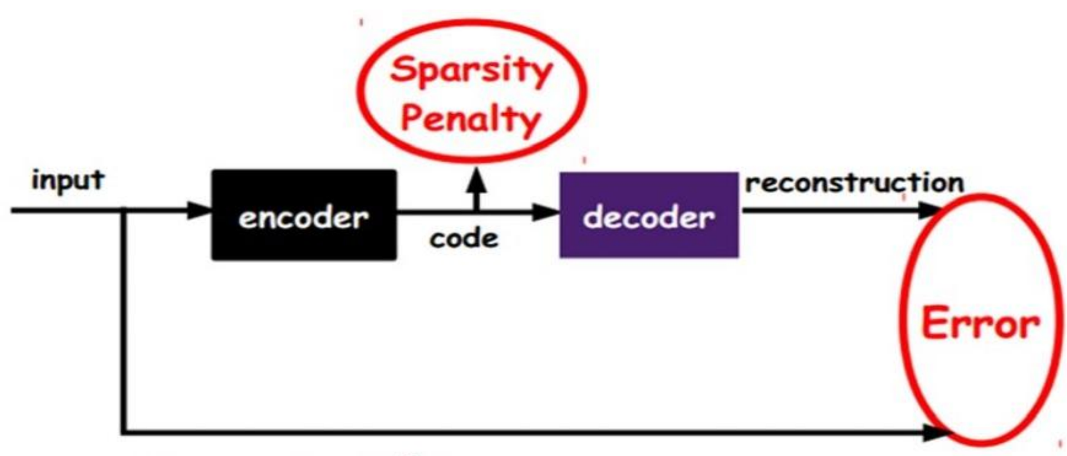


图 7.34 稀疏自动编码

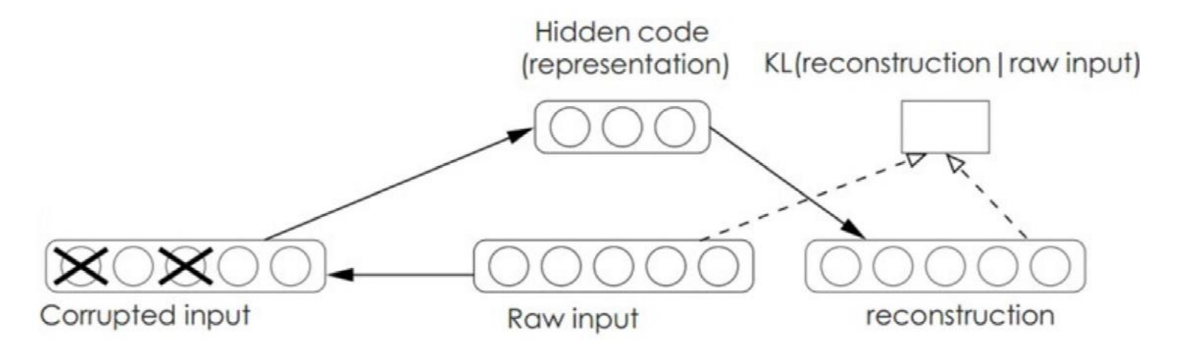


图 7.35 降噪自动编码

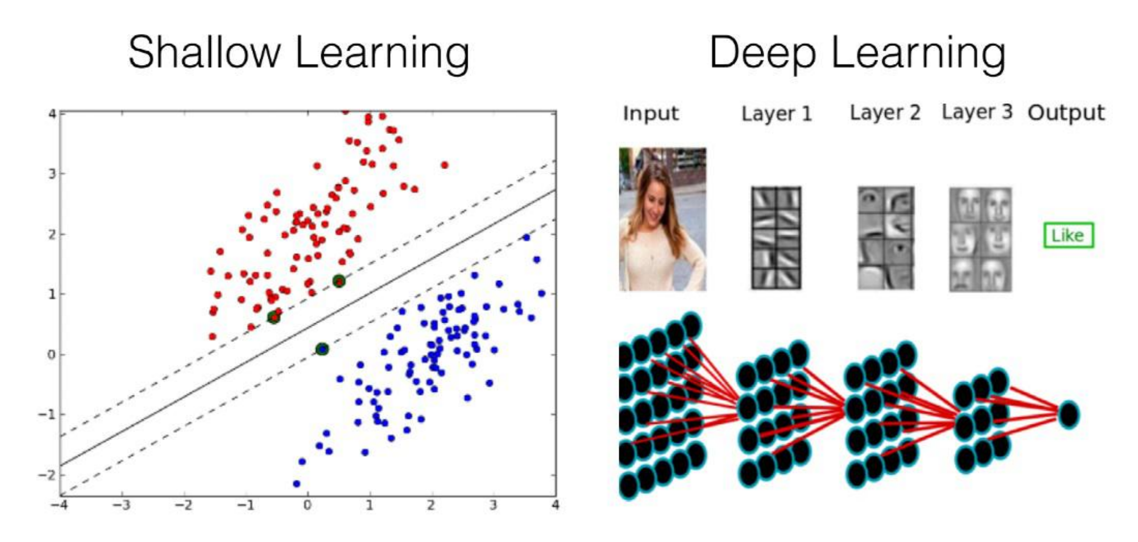


图 7.36 从浅层走向深度学习（1）

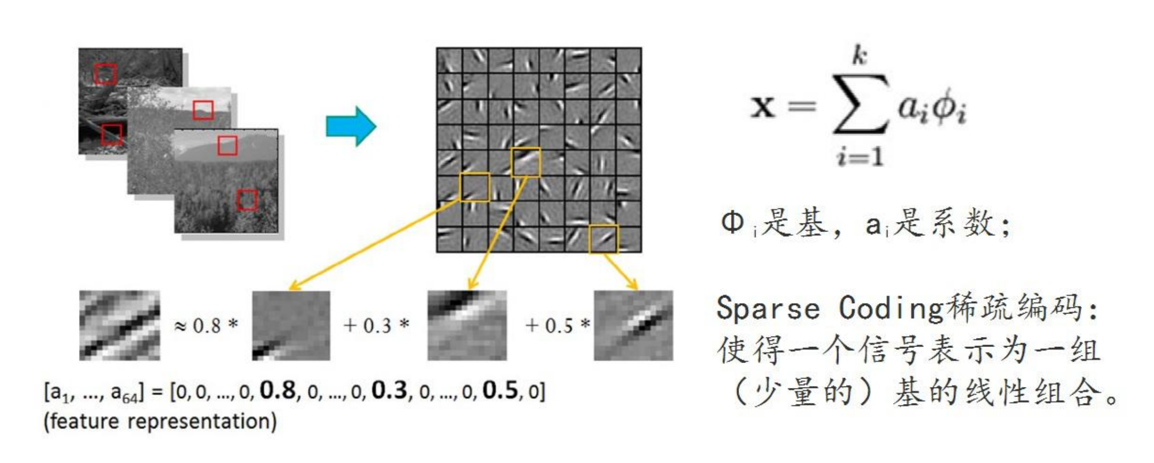


图 7.37 从浅层走向深度学习（2）

Sparse Coding的训练阶段

$$\min_{a,\phi} \sum_{i=1}^{m} \left\| x_i - \sum_{j=1}^{k} a_{i,j}\phi_j \right\|^2 + \lambda \sum_{i=1}^{m}\sum_{j=1}^{k} |a_{i,j}|$$

Sparse Coding编码阶段

$$\min_{a} \sum_{i=1}^{m} \left\| x_i - \sum_{j=1}^{k} a_{i,j}\phi_j \right\|^2 + \lambda \sum_{i=1}^{m}\sum_{j=1}^{k} |a_{i,j}|$$

图 7.38 从浅层走向深度学习（3）

从浅层走向深度学习的一个关键，是需要 RBM 作为节点，进行判断，使得数值能够各层神经网络，最终指向输出层。

限制波尔兹曼机（RBM），设所有的节点都是随机二值变量节点（只能取 0 或者 1 值），并且全概率分布 p(v,h)满足 Boltzmann 分布。

其公式为：【 p(h|v)=p(h1|v)···p(hn|v) 】

该公式的含义是：在已知 v 的情况下，所有的隐藏节点之间是条件独立（因节点之间不存在连接）。

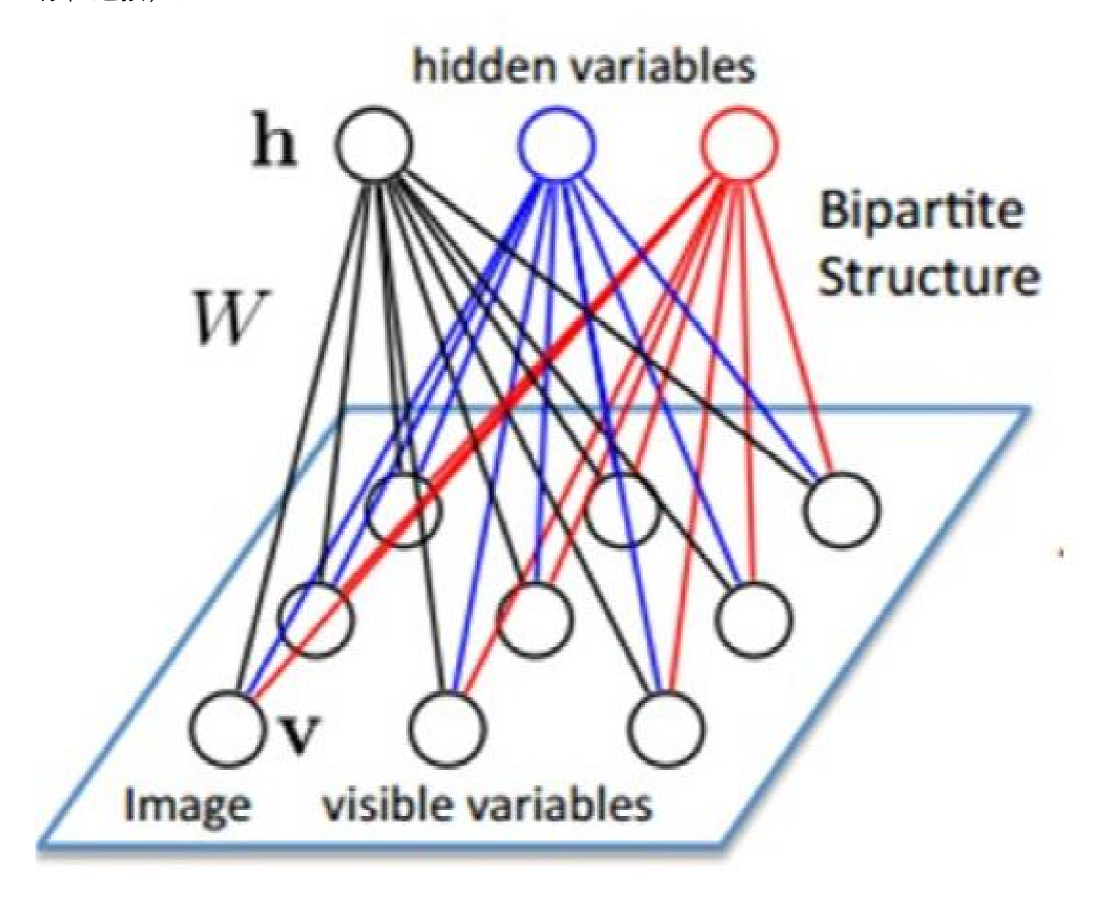


图 7.39 限制波尔兹曼机（1）

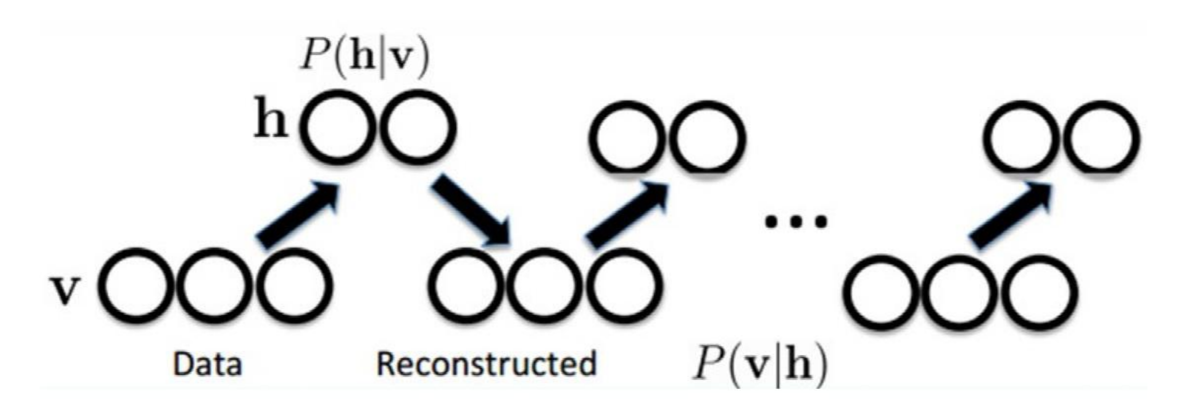


图 7.40 限制波尔兹曼机（2）

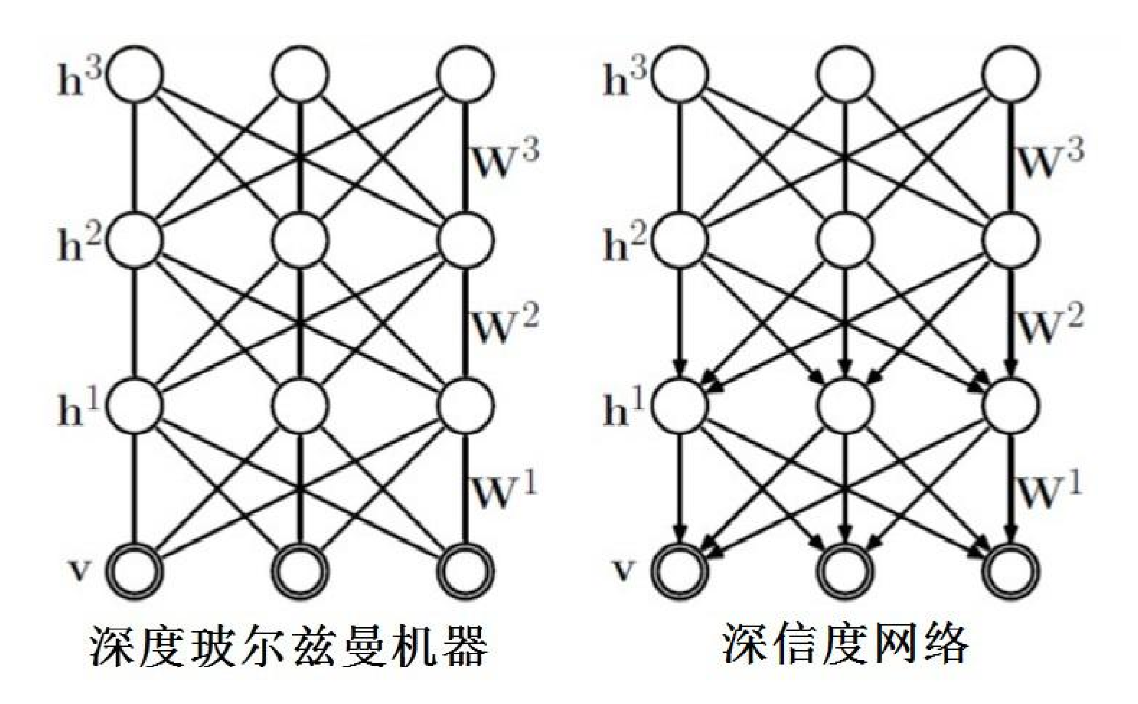


图 7.41 限制波尔兹曼机（3）

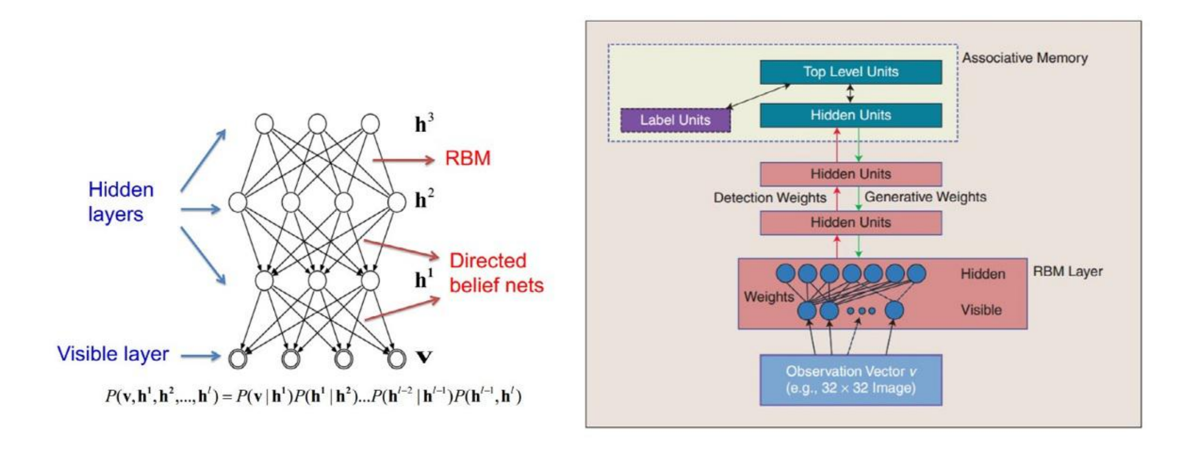


图 7.42 深信度网络

深度学习主要引入两个思想：

（一）分层判断的无监督学习（layer-wise unsupervised learning）

（二）有监督的长期微调（Supervised fine-turning）

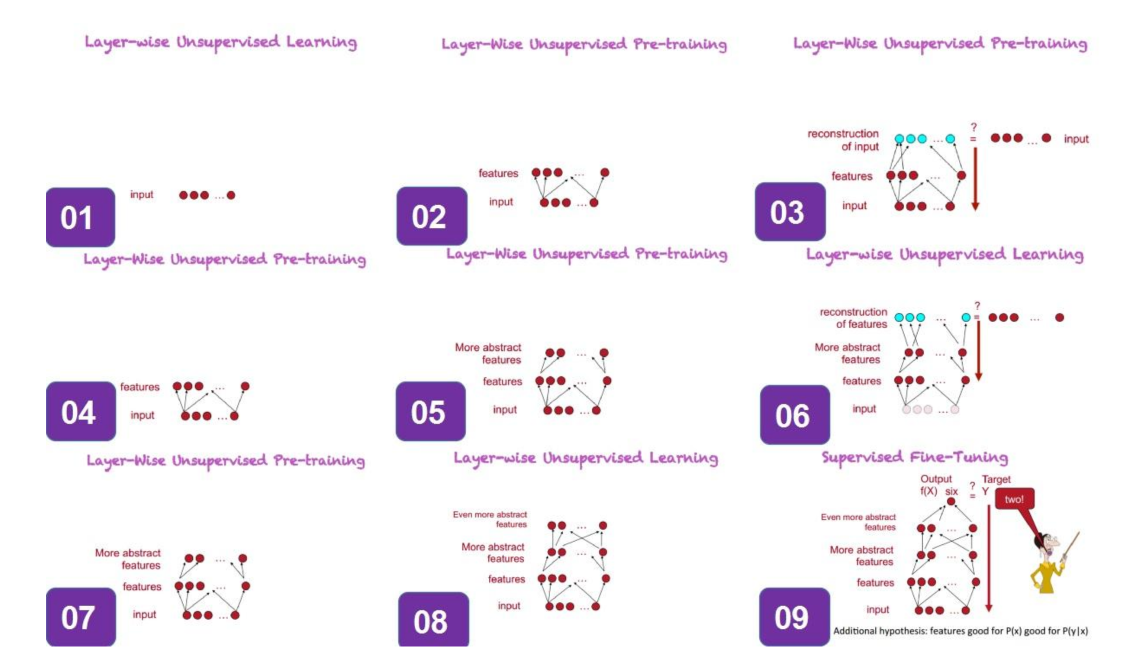


图 7.43 深度学习的无监督分层学习和有监督微调学习（Bengio）

四、软件操作

如前面第一部分所述，深度学习改变我们的生活以及科研方式，其实现工具之一 TensorFlow 有几个经典应用，摘要简述如下：

（一）Gmail、Play Recommendation、Search、Translate、Map 等；

（二）科学家用 TF 来搭建基于视网膜进行预防糖尿病致盲的研究；

（三）TF 应用在音乐、绘画上，构建深度学习模型；

（四）TF 框架与传感器构建自动化的海洋生物检测系统；

（五）TF 在移动客户端上进行翻译、风格化等工作；

（六）TF 在移动设备 CPU（高通 820）上，达到更高性能和更低功耗；

（七）TensorBoard Embedded vector 可视化工作等。

那么，什么是 TensorFlow 呢？

它的命名，来源于它本身的原理：Tensor（张量）意味着 N 维数组，Flow（流）意味着基于数据流图的计算。Tensorflow 运行过程就是张量从图的一端流动到另一端的计算过程。TensorFlow 是一个采用数据流图（data flow graphs），用于数值计算的开源软件库。最初由 Google 大脑小组开发出来，用于机器学习和深度神经网络方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。它是谷歌基于 DistBelief 进行研发的第二代人工智能学习系统。2015 年 11 月 9 日，Google 发布人工智能系统 TensorFlow 并宣布开源。

TensorFlow 的基本要素有五：

（一）图（Graph）：描述计算过程，TF 使用图来表示计算任务。

（二）张量（Tensor）：TF 使用 tensor 表示数据。每个 Tensor 是一个类型化的多维数组。

（三）操作（op）：图中的节点被称为 op（opearation），一个 op 获得 0 个或多个 Tensor，执行计算，产生 0 个或多个 Tensor。

（四）会话（Session）：图必须在称之为“会话”的上下文中执行。会话将图的 op 分

发到诸如 CPU 或 GPU 之类的设备上执行。

（五）变量（Variable）：运行过程中可以被改变，用于维护状态。

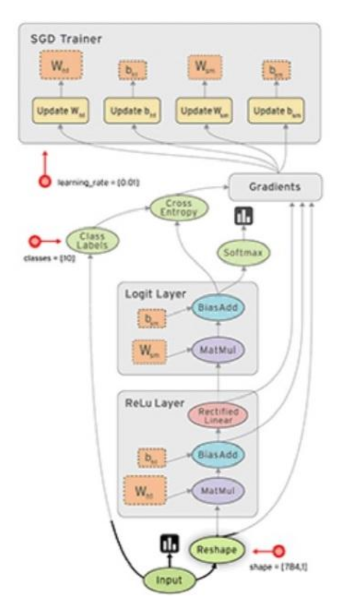
TensorFlow 使用计算图来表示一个计算任务。图中的节点代表数学运算，也可 以表示数据的输入、输出和读写等操作；图中的边表示多维数组(Tensors）节点之间的某种联系。TensorFlow 使用计算图来构建计算过程，用符号来表示计算操作。这使得 TensorFlow 可以同时运用 Linux 64 位操作系统的 CPU 和 GPU 性能，也可以在移动端 Android 或者 iOS 上执行。

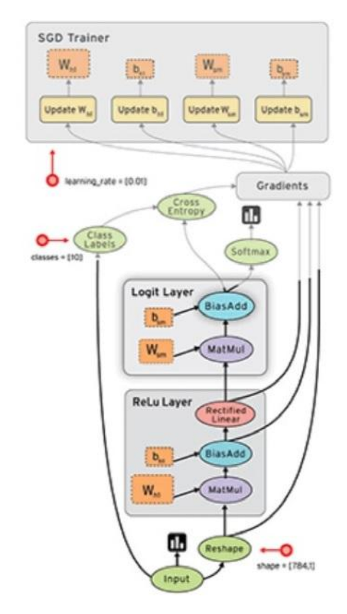
数据流图（Data Flow Graph）用“节点”(nodes)和“线”（edges）的有向图来描述数学计算。

（一）“节点”表示施加的数学操作，但也可以表示数据输入（feed in）的起点/输出（push out）的终点，或者是读取/写入持久变量（persistent variable）的终点。

（二）“线”表示“节点”之间的输入/输出关系。

这些数据“线”可以运输“size 可动态调整”的多维数组，即“张量”（tensor）。





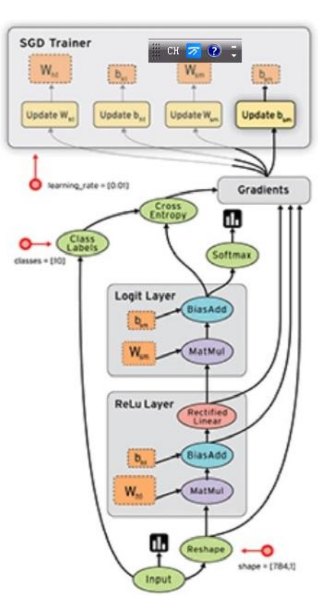


图 9.44 数据流图

TensorFlow 的特性

（一）灵活性：TensorFlow 不是一个严格的“神经网络”库。只要将计算表示为一个数据流图，就可以使用 TensorFlow。

（二）可移植性（Portability）：Tensorflow 可运行在台式机、服务器、手机移动等设备上，而且在多 CPU 和多 GPU 上运行。

（三）多语言支持：Tensorflow 提供了一套易用的 Python 使用接口来构建和执行 graphs 并且易于 C++使用的接口，未来还会支持 Go、Java、Lua、JavaScript、R 等。

在 Linux 环境安装 TensorFlow 的方式

（一）用 Pip install 安装，可能会升级您之前安装过的 Python 包，对您机器上的 Python 程序造成影响。

（二）用 Anaconda install 安装，这种方式会把 TensorFlow 安装在 Anaconda 提供的环境中，不会影响其他 Python 程序，但是 Anaconda 本身占据存储空间和内存资源。

（三）采取 Installing from sources 方式（从源码进行安装），是把 TensorFlow 源码构建成一个 pip wheel 文件，使用 pip 工具安装它。

为了更为直观理解深度学习，我们先从 Tensorflow 入门，用最简单的方式，实现 Tensorflow 第一个程序。再逐步掌握工具之后，再一层层从头到脚理解深度学习。

```
import tensorflow as tf
a = tf.placeholder("float")
b = tf.placeholder("float")
y = tf.mul(a, b)
## 导入 Python 模块 tensorflow
## 定义符号变量（也称为占位符）。在后面程序执行中会操作这些变量。
## 此处，把这些变量作为参数，然后 TensorFlow 的乘法函数 tf.mul 会调用之前定义的变量。数学函数 tf.mul 操作 tensor 动态处理大小数值、多维数组等。

sess = tf.Session()
print sess.run(y, feed_dict={a:3,b:3})
```

TensorFlow 的算数操作符

tf.add，tf.sub，tf.mul，tf.div，tf.mod，tf.abs，tf.neg，tf.sign，tf.inv，tf.square，tf.round，tf.sqrt，tf.pow，tf.exp，tf.log，tf.maximum，tf.minimum，tf.cos，tf.sin。

表 7.1 TensorFlow 的数学函数

指令	描述（作用）
tf.diag	给定对角线上的值，返回对角 tensor
tf.transpose	转置
tf.matmul	tensor 乘法，即矩阵乘法
tf.matrix_determinant	方阵的行列式
tf.matrix_inverse	方阵的逆矩阵

表 7.2 TensorFlow 在 GPU 下的核

操作（核）	操作分组
Maths	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal
Array	Concat, Slice, Split, Constant, Rank, Shape, Shuffle
Matrix	MatMul, MatrixInverse,MatrixDeterminant
Neuronal Network	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool
Checkpointing	Save, Restore
Queues and syncronizations	Enqueue, Dequeue, MutexAcquire, MutexRelease
Flow control	Merge, Switch, Enter, Leave, NextIteration

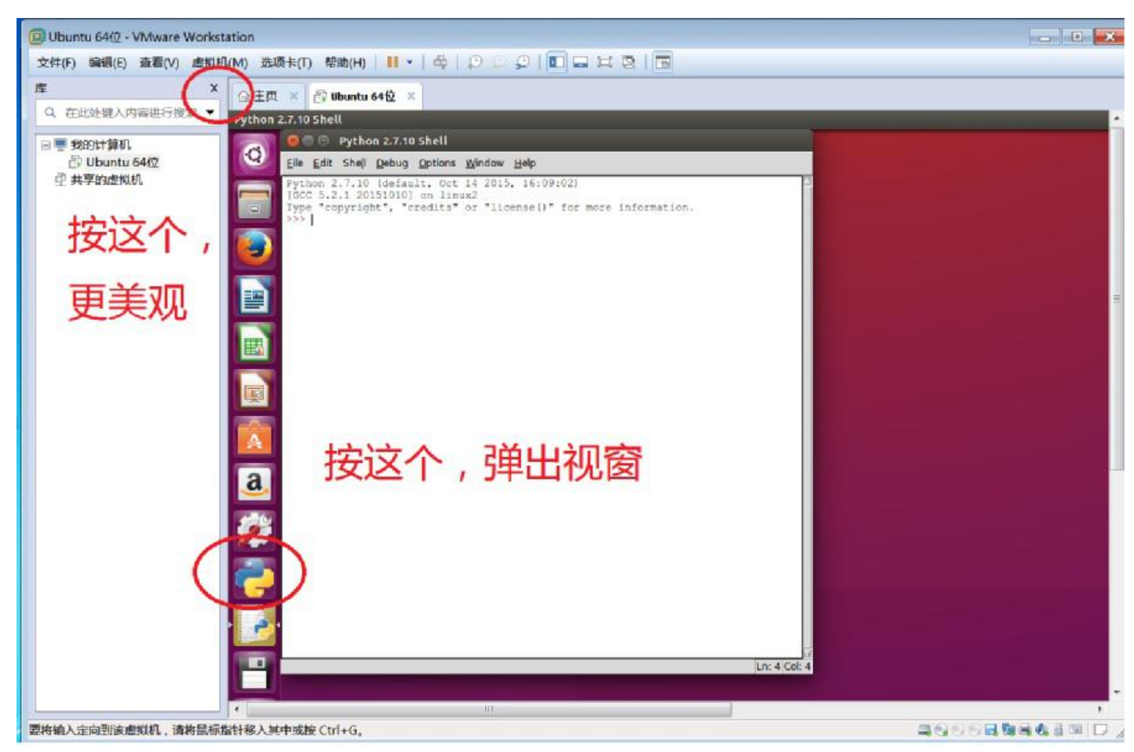


图 7.45 实现第一个 TensorFlow 程序（1）

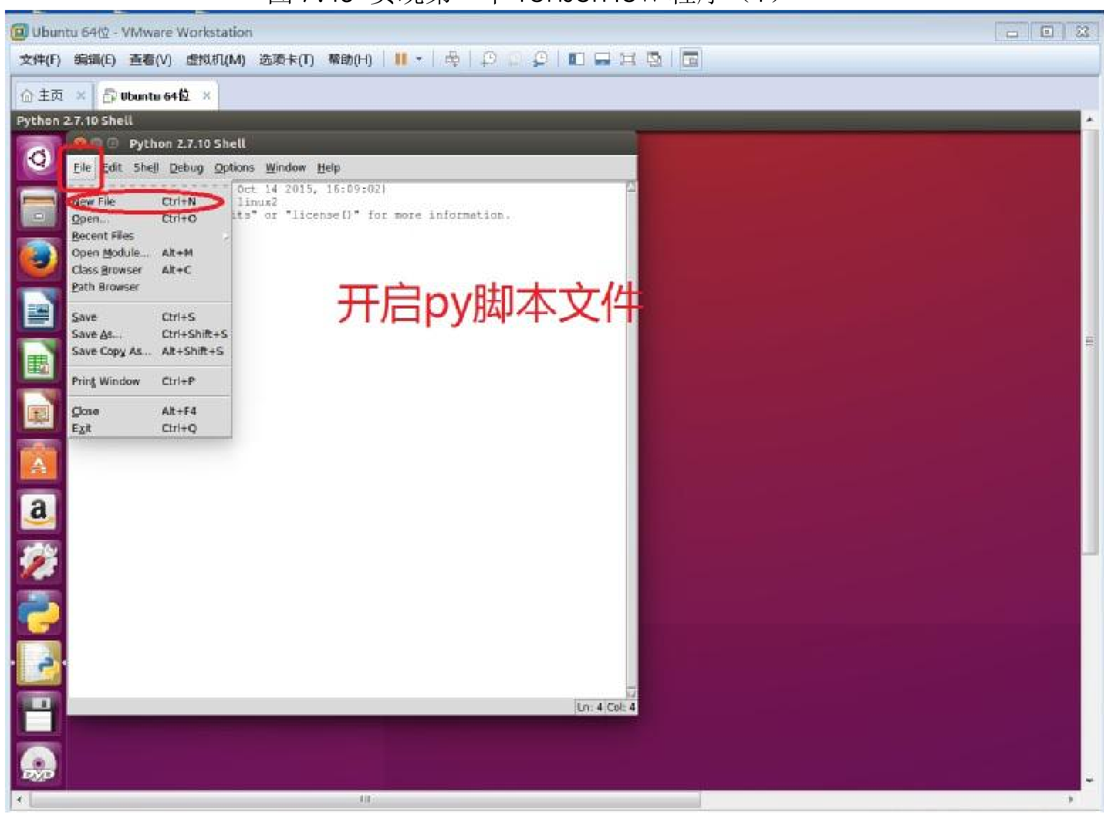


图 7.46 实现第一个 TensorFlow 程序（2）

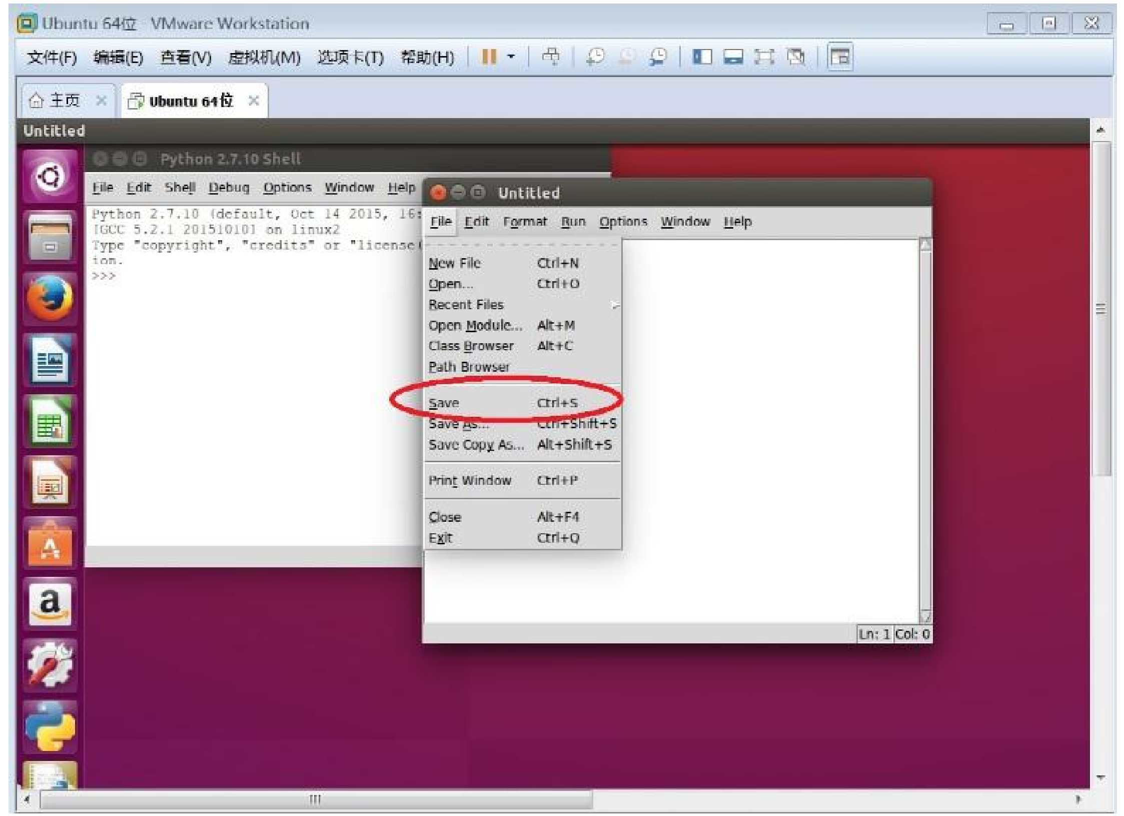


图 7.47 实现第一个 TensorFlow 程序（3）

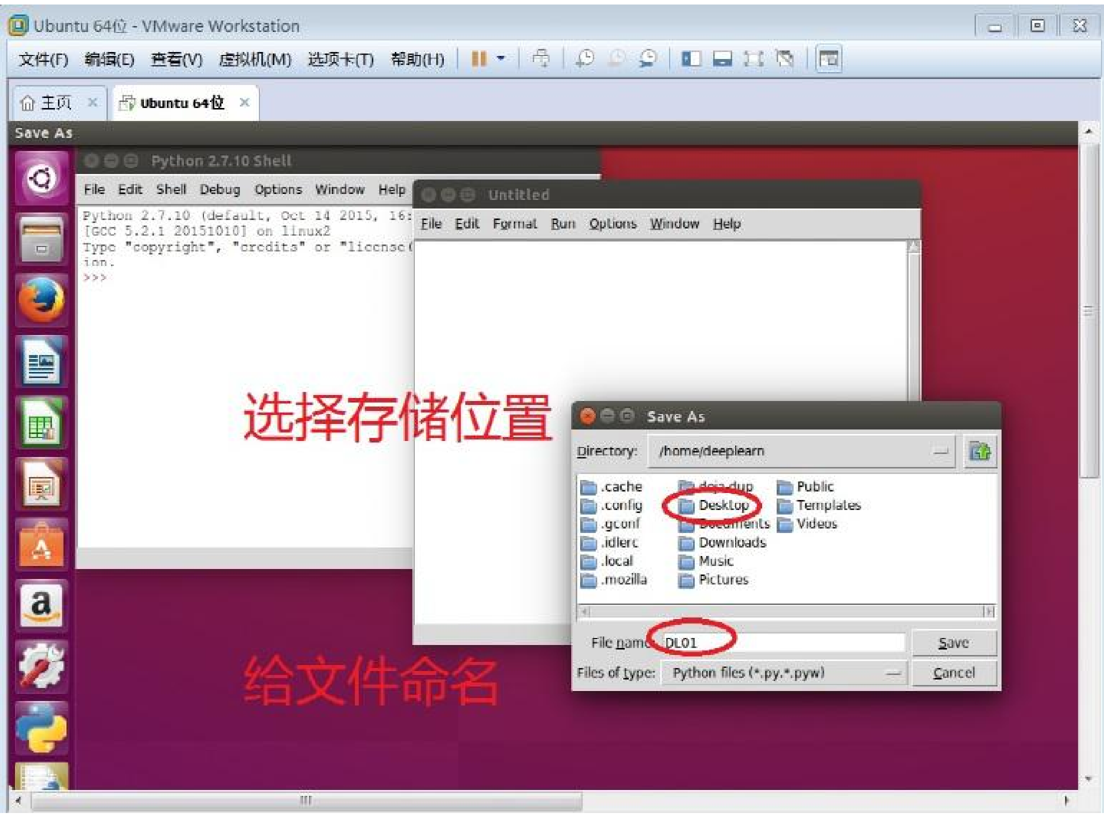


图 7.48 实现第一个 TensorFlow 程序（4）

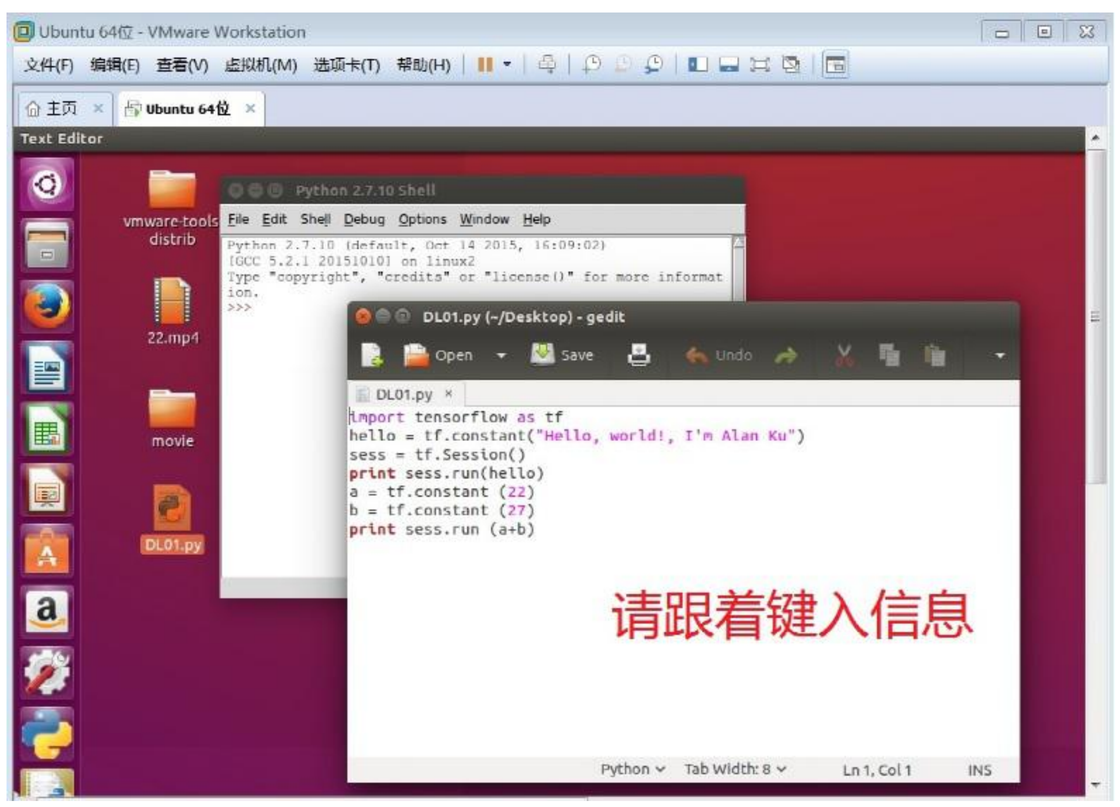


图 7.49 实现第一个 TensorFlow 程序（5）

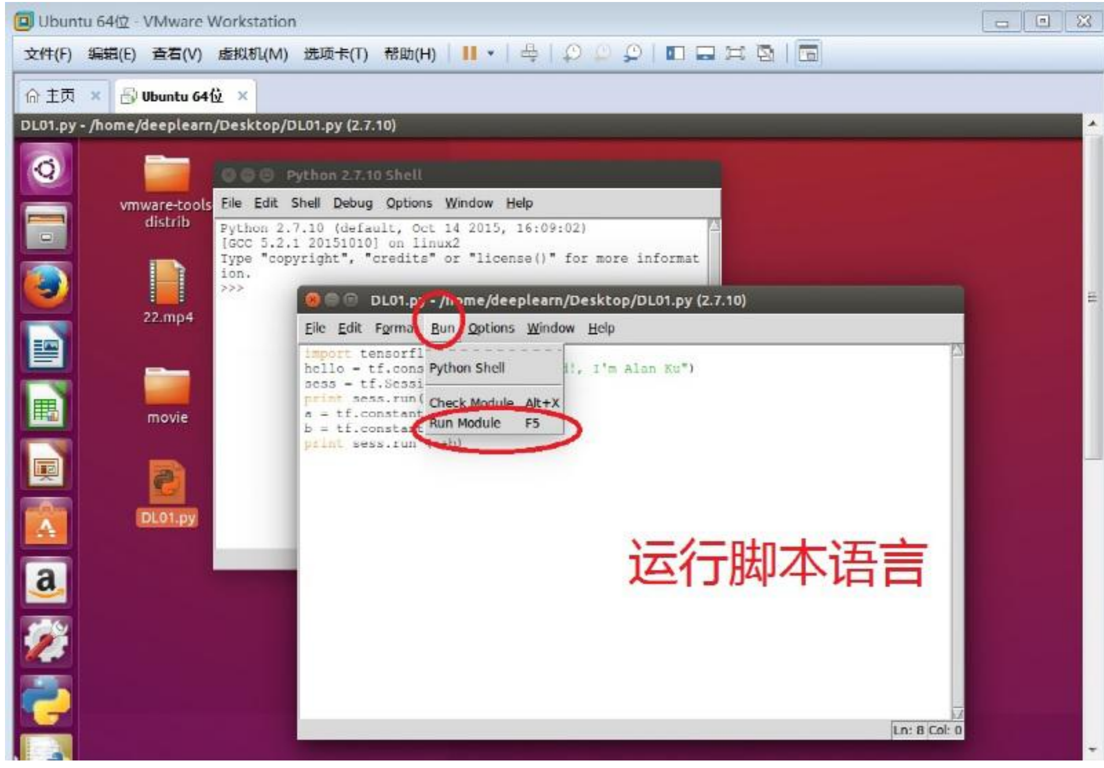


图 7.50 实现第一个 TensorFlow 程序（6）

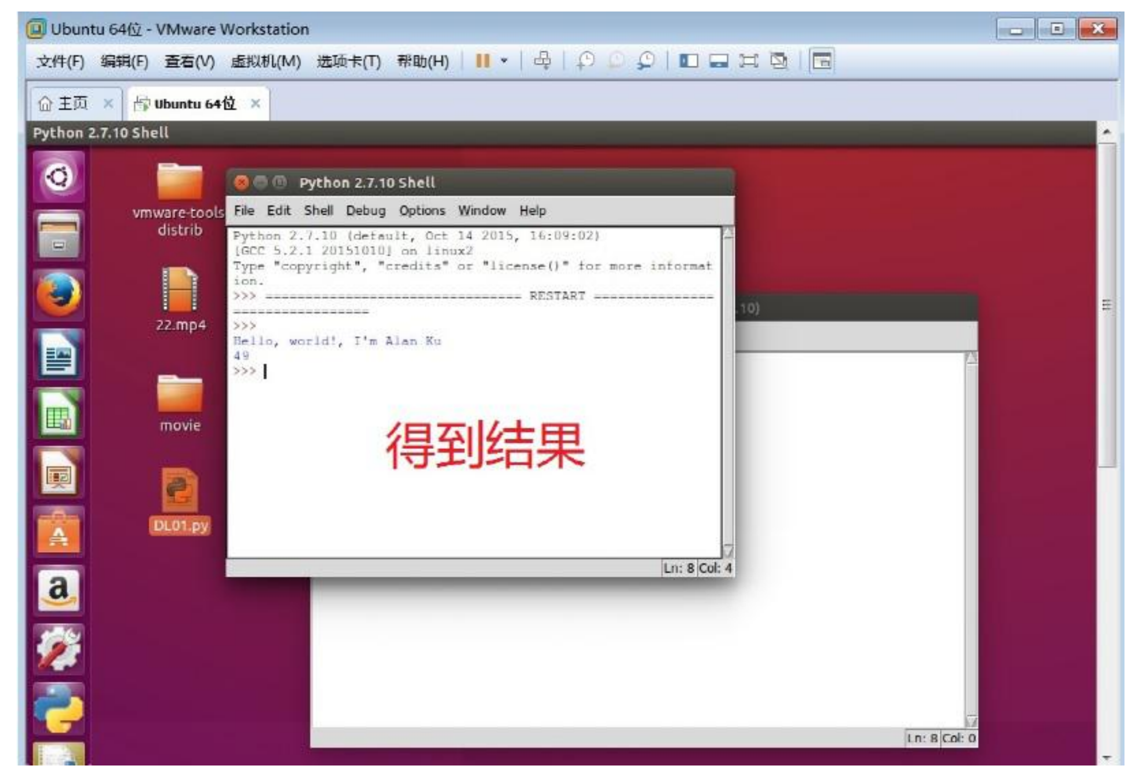


图 7.51 实现第一个 TensorFlow 程序（7）

在我们实现第一个 TensorFlow 程序之后，我们可以稍微扩展一下，用一个简短流程，把 TF 进行整体掌握。

（一）首先导入 TensorFlow 建立常数；

（二）接着，把常数存储在张量（Tensors）里进行定义；

（三）建立 Tensorflow 会话（Session）进行计算，这个步骤与刚刚练习的内容相似；

（四）紧接着，我们采用对话模式 tf.InteractiveSession()工作；

（五）对于日后 TF 的操作而言，接下来的任务更为重要，首先是“更改权重”。

（六）接着，初始化后进行计算；

（七）最后，在任意节点检查数值或者定义数值。

完成以上七个步骤的练习之后，我们就可以进入第二节的练习，因为所要关注的不再是 TF 的编程内容，而是具体的神经网络的内容，在第二节打好基础之后，就第三节深度学习的学习就会显得容易许多。

导入 TensorFlow 建立常数，在 Python Shell 的代码如下：

```
import tensorflow as tf
a = tf.constant(1)
b = tf.constant(2)
c = a + b
d = a * b
```

结果：

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: import tensorflow as tf

In [2]: a = tf.constant(1)

In [3]: b = tf.constant(2)

In [4]: c = a + b

In [5]: d = a * b

In [6]:
```

图 7.52 导入 TensorFlow 建立常数

把常数存储在张量（Tensors）里进行定义，在 Python Shell 的代码如下：

```
V1 = tf.constant([1., 2.])  #一维向量
V2 = tf.constant([3., 4.])  #一维向量
M = tf.constant([[1., 2.]]) #二维矩阵
N = tf.constant([[1., 2.],[3.,4.]]) #二维矩阵
K = tf.constant([[[1., 2.],[3.,4.]]])  #三维矩阵
V3 = V1 + V2    #进行计算，注意向量边角（Shape）。
M2 = M * M  #进行计算，默认是元素级（Element）操作。
NN = tf.matmul(N,N) # 矩阵乘法需要 tf.matmul( ,)调用。
```

结果：

```
In [6]: V1 = tf.constant([1., 2.])

In [7]: V2 = tf.constant([3., 4.])

In [8]: M = tf.constant([[1., 2.]])

In [9]: N = tf.constant([[1., 2.],[3.,4.]])

In [10]: K = tf.constant([[[1., 2.],[3.,4.]]])

In [11]: V3 = V1 + V2

In [12]: M2 = M * M

In [13]: NN = tf.matmul(N,N)

In [14]:
```

图 7.53 把常数存储在张量（Tensors）里进行定义

建立 Tensorflow 会话（Session）进行计算，在 Python Shell 的代码如下：

```
sess = tf.Session()     #建立 TF 会话，而不只是 TF 定义。
```

```
output = sess.run(NN)    #进行上述 NN（矩阵相乘）的计算。
print("NN is:")      #输出结果。
 print(output)
sess.close()    #结束会话
```

结果：

```
In [14]: sess = tf.Session()

In [15]: output = sess.run(NN)

In [16]: print("NN is:")
NN is:

In [17]: print("NN is:")
    ...: print(output)
NN is:
[[ 7. 10.]
 [15. 22.]]

In [18]: sess.close()

In [19]:
```

图 7.54 建立 Tensorflow 会话（Session）进行计算

采用对话模式 tf.InteractiveSession()工作，在 Python Shell 的代码如下：

```
sess = tf.InteractiveSession()
print("M2 is:")
print(M2.eval())
```

结果：

```
In [18]: sess.close()

In [19]: sess = tf.InteractiveSession()

In [20]: print("M2 is:")
    ...: print(M2.eval())
M2 is:
[[1. 4.]]

In [21]:
```

图 7.55 采用对话模式 tf.InteractiveSession()工作

更改权重，在 Python Shell 的代码如下：

```
W = tf.Variable(0, name="weight")
```

```
init_op = tf.initialize_all_variables() #初始化 varialbe
sess.run(init_op)
print("W is:")
print(W.eval())
## TF 的 variable 更改数值后，就变更权重了，但是 variable 需要在使用前进行初始化。
```

结果：

```
In [21]: W = tf.Variable(0, name="weight")
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow
\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.

In [22]: init_op = tf.initialize_all_variables()
    ...: sess.run(init_op)
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow
\python\util\tf_should_use.py:193: initialize_all_variables (from
tensorflow.python.ops.variables) is deprecated and will be removed after
2017-03-02.
Instructions for updating:
Use `tf.global_variables_initializer` instead.

In [23]: print("W is:")
    ...: print(W.eval())
W is:
0
```

图 7.56 更改权重

初始化后进行计算，在 Python Shell 的代码如下：

```
W += a
print("W after adding a:")
print(W.eval())

W += a
print("W after adding a again:")
print(W.eval())
```

结果：

```
In [24]: W += a
    ...: print("W after adding a:")
    ...: print(W.eval())
WARNING:tensorflow:Variable += will be deprecated. Use variable.assign_add if
you want assignment to the variable value or 'x = x + y' if you want a new
python Tensor object.
W after adding a:
1

In [25]: W += a
    ...: print("W after adding a again:")
    ...: print(W.eval())
W after adding a again:
2
```

图 7.57 初始化后进行计算

在任意节点检查数值或者定义数值，在 Python　Shell 的代码如下：

```
E = d + b #实际上，这里的计算是 1*2+2 所以=4
print("E as defined:")
print(E.eval())

print("E and d:") #查看同一时间点的 d 的情况。
print(sess.run([E,d]))

print("E with custom d=4:")
print(sess.run(E, feed_dict = {d:4.}))
# 通过指定【字典】定义 d=4
```

结果：

```
In [26]: E = d + b

In [27]: print("E as defined:")
    ...: print(E.eval())
E as defined:
4

In [28]: print("E and d:")
    ...: print(sess.run([E,d]))
E and d:
[4, 2]

In [29]: print("E with custom d=4:")
    ...: print(sess.run(E, feed_dict = {d:4.}))
E with custom d=4:
6

In [30]:
```

图 7.58 在任意节点检查数值或者定义数值

第二节 基础知识

一、最小二乘法 OLS 回归

此处，我们回到了第四章数据统计的延续，讨论的是线性回归里的一个特殊问题，我们甚至不讨论第五章数据建模里的交叉检验以及第六章机器学习里的训练集和测试集的拆分方式，我们此处讨论线性回归的问题，是为了 9.2.2 的罗吉斯回归进行比较。

普通最小二乘法或 OLS 也可以称为线性最小二乘。这是一种用于近似确定位于线性回归模型中的未知参数的方法。通过最小化数据集内观察到的响应与线性近似预测的响应之间的垂直距离平方的总和，可以得到普通最小二乘。

在线性回归中，最小二乘法就是找到一条直线，使所有样本到直线的“欧式距离和”最小。从理论上说，数据越多，效果越好，即所估计的直线方程越能更好地反映变量之间的关系。最小二乘法可以帮助我们在进行线性拟合时，如何选择“最好”的直线。

第一步，导入工具。

```
# 导入三个主要的库
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# 事先安装 statsmodels 库，安装时可能会有错误，多试试几次。
from statsmodels.formula.api import ols
# 导入：线性模型 de 方差分析 de 模块
from statsmodels.stats.api import anova_lm
# 导入：交叉效应和线段绘图模块
from statsmodels.graphics.api import interaction_plot, abline_plot
```

第二步，准备数据。

```
# 读取数据，数据集 barley 是不同品种的大麦在不同地区的产量数据。
b = pd.read_csv('barley.csv')
print(b.head())
## 第一列是序列号，没有什么意义，需要删除。

# 删除第一列，那个 id 列。删除行是 axis=0（默认值），删除列要=1 才行。
b = b.drop(b.columns[[0]], axis = 1)

# 数据处理后的数据集的前六条（行）数据。
print(b.head())
# 变量名称
print(b.columns)
# 数据形状
print(b.shape)
```

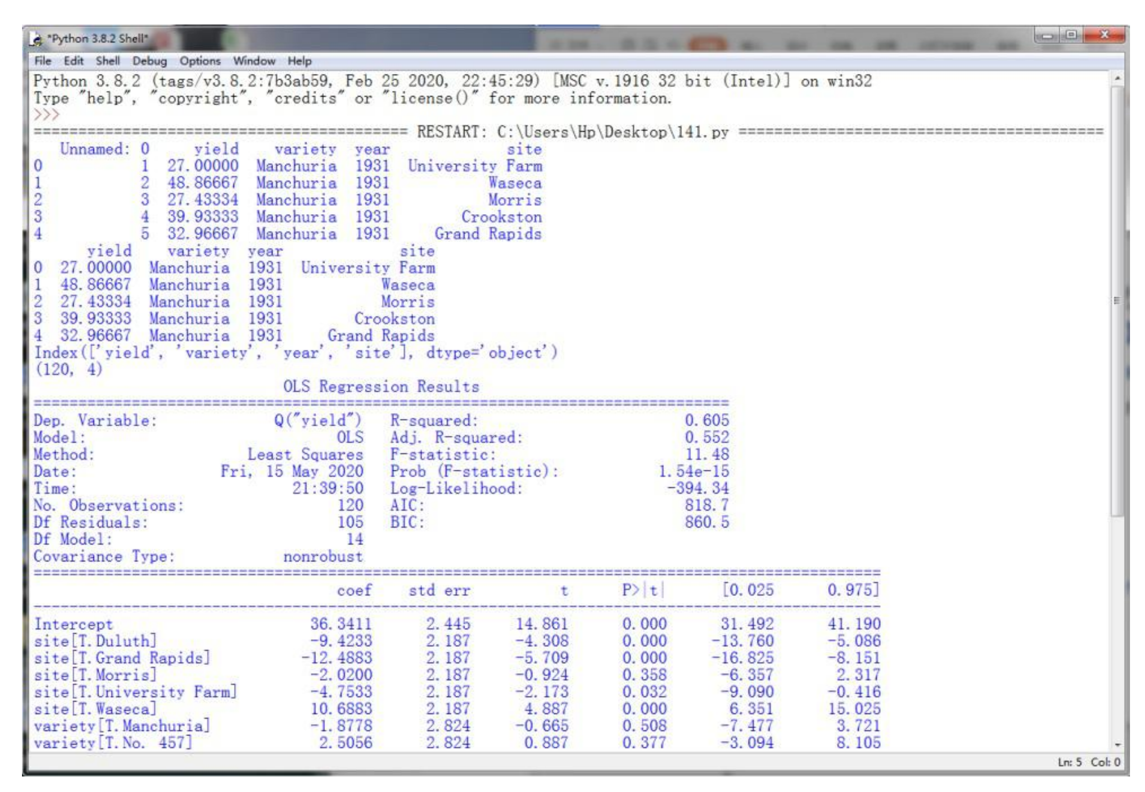
变量，包括：产量 yield、品种 variety(有 10 种)、地点 site（有 6 处）、年份 year（有 2 年）。产量、年份，这两个变量，属于：定量变量，但是产量是等比变量（四则运算皆可），

年份是等差变量（乘除没有意义），它们的数据类型是连续性数值。地点、品种，这两个变量，属于：定性变量，用作分类，又可称为分类变量，它们的数据类型是字符串数值。

第三步，建立模型。

```
# 建立【拟合带有交叉项次】的线性模型
# 建立 Ordinary least squares (OLS)模型。
mod = ols(formula='Q("yield")~site+variety',data= b)
# 模型拟合 fit()
lm = mod.fit()
# 模型的概况的信息
print(lm.summary())
## 标准误差（Standard Errors）假定，误差的协方差矩阵（the covariance matrix of the errors）被正确指定。
```

结果：



```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:\Users\Hp\Desktop\141.py ====================
   Unnamed: 0      yield    variety  year             site
0           1   27.00000  Manchuria  1931  University Farm
1           2   48.86667  Manchuria  1931           Waseca
2           3   27.43334  Manchuria  1931           Morris
3           4   39.93333  Manchuria  1931        Crookston
4           5   32.96667  Manchuria  1931     Grand Rapids
      yield    variety  year             site
0  27.00000  Manchuria  1931  University Farm
1  48.86667  Manchuria  1931           Waseca
2  27.43334  Manchuria  1931           Morris
3  39.93333  Manchuria  1931        Crookston
4  32.96667  Manchuria  1931     Grand Rapids
Index(['yield', 'variety', 'year', 'site'], dtype='object')
(120, 4)
                            OLS Regression Results
==============================================================================
Dep. Variable:             Q("yield")   R-squared:                       0.605
Model:                            OLS   Adj. R-squared:                  0.552
Method:                 Least Squares   F-statistic:                     11.48
Date:                Fri, 15 May 2020   Prob (F-statistic):           1.54e-15
Time:                        21:39:50   Log-Likelihood:                -394.34
No. Observations:                 120   AIC:                             818.7
Df Residuals:                     105   BIC:                             860.5
Df Model:                          14
Covariance Type:            nonrobust
===========================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
Intercept                  36.3411      2.445     14.861      0.000      31.492      41.190
site[T.Duluth]             -9.4233      2.187     -4.308      0.000     -13.760      -5.086
site[T.Grand Rapids]      -12.4883      2.187     -5.709      0.000     -16.825      -8.151
site[T.Morris]             -2.0200      2.187     -0.924      0.358      -6.357       2.317
site[T.University Farm]    -4.7533      2.187     -2.173      0.032      -9.090      -0.416
site[T.Waseca]             10.6883      2.187      4.887      0.000       6.351      15.025
variety[T.Manchuria]       -1.8778      2.824     -0.665      0.508      -7.477       3.721
variety[T.No. 457]          2.5056      2.824      0.887      0.377      -3.094       8.105
```

图 7.59 建立【拟合带有交叉项次】的线性模型

我们可以在 Python shell 里，键入如下指令，而不必完全依靠.py 文件，来查看数据以及模型的情况。

```
# 第一列是序列号，没有什么意义，需要删除。
print(b.head())

# 数据形状
print(b.shape)

# 模型的概况的信息
print(lm.summary())
```

第四步，方差分析表。

```
# 添加图形的标签、题名、图框边长以及图例
plt.xlabel("x")
# 累积分布函数(Cumulative Distribution Function)
plt.ylabel("CDF(x)")
# 逻辑回归方程
plt.title("Logistic function")
plt.ylim(ymin=-0.05, ymax=1.05)
plt.xlim(xmin=-10.05, xmax=10.05)
plt.legend(["Scale=%.2f" % scale for scale in scales], loc="lower right")

# 产生设计矩阵
lm.model.exog[:5]
lm.model.data.orig_exog[:5]
# 产生方差分析并且绘制表格
table = anova_lm(lm)
print(table)
```

结果：

```
*Python 3.8.2 Shell*
File Edit Shell Debug Options Window Help
Model:                            OLS   Adj. R-squared:                  0.552
Method:                 Least Squares   F-statistic:                     11.48
Date:                Fri, 15 May 2020   Prob (F-statistic):           1.54e-15
Time:                        21:39:50   Log-Likelihood:                -394.34
No. Observations:                 120   AIC:                             818.7
Df Residuals:                     105   BIC:                             860.5
Df Model:                          14
Covariance Type:            nonrobust
==================================================================================================
                                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------
Intercept                         36.3411      2.445     14.861      0.000      31.492      41.190
site[T.Duluth]                    -9.4233      2.187     -4.308      0.000     -13.760      -5.086
site[T.Grand Rapids]             -12.4883      2.187     -5.709      0.000     -16.825      -8.151
site[T.Morris]                    -2.0200      2.187     -0.924      0.358      -6.357       2.317
site[T.University Farm]           -4.7533      2.187     -2.173      0.032      -9.090      -0.416
site[T.Waseca]                    10.6883      2.187      4.887      0.000       6.351      15.025
variety[T.Manchuria]              -1.8778      2.824     -0.665      0.508      -7.477       3.721
variety[T.No. 457]                 2.5056      2.824      0.887      0.377      -3.094       8.105
variety[T.No. 462]                 2.0361      2.824      0.721      0.472      -3.563       7.635
variety[T.No. 475]                -1.5806      2.824     -0.560      0.577      -7.180       4.019
variety[T.Peatland]                0.8389      2.824      0.297      0.767      -4.760       6.438
variety[T.Svansota]               -2.9639      2.824     -1.050      0.296      -8.563       2.635
variety[T.Trebi]                   6.0583      2.824      2.145      0.034       0.459      11.657
variety[T.Velvet]                 -0.2806      2.824     -0.099      0.921      -5.880       5.319
variety[T.Wisconsin No. 38]        6.0528      2.824      2.143      0.034       0.454      11.652
==============================================================================
Omnibus:                        8.544   Durbin-Watson:                   2.014
Prob(Omnibus):                  0.014   Jarque-Bera (JB):                3.512
Skew:                           0.055   Prob(JB):                        0.173
Kurtosis:                       2.169   Cond. No.                         11.6
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
             df       sum_sq      mean_sq          F        PR(>F)
site        5.0  6633.853084  1326.770617  27.731803  8.321706e-18
variety     9.0  1052.571814   116.952424   2.444508  1.442913e-02
Residual  105.0  5023.507265    47.842926        NaN           NaN
|
                                                                    Ln: 61  Col: 0
```

图 7.60 方差分析表

第五步，交叉效应图。

```
# 绘制交叉效应图
interaction_plot(b['site'],b['variety'], b['yield'], legendloc=2)
##前两位是定性变量，最后一位是定量变量，才能看出交叉效应。
##此处 X 轴是地点，图上标识是品种，Y 轴是产量，用来探索地点和品种所产生的产量效应。
plt.show()
```

结果：

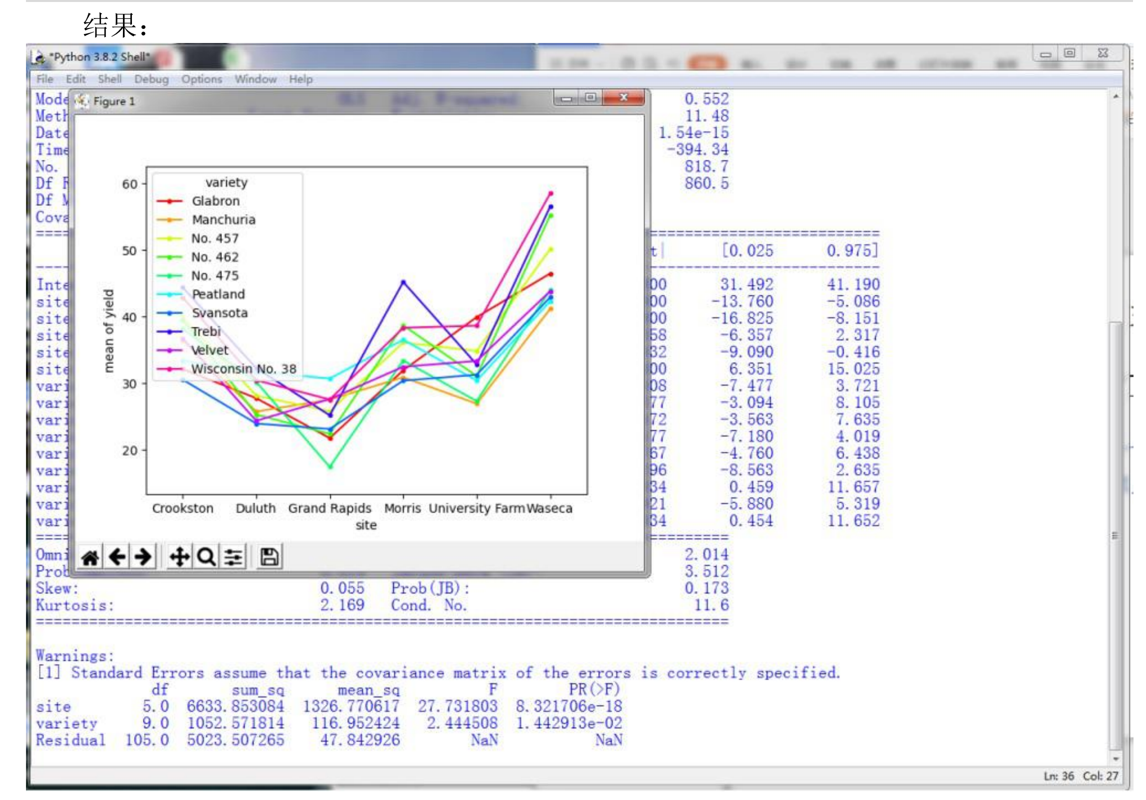


图 7.61 交叉效应图

最大似然估计或 MLE 是一种用于估计统计模型的参数以及将统计模型拟合到数据的方法。线性回归用 MLE 求解参数，最小二乘线性回归也是由 MLE 推导而来。与最小二乘法回归相比，我们接下来对比 logitic 回归。这个 logistic 回归用来分类，它是进入多层神经网络的一个重要知识和技术。

二、罗吉斯回归

与最小二乘线性回归不同，罗吉斯回归是神经网络的一个重要组成部分，而神经网络是深度学习的一个重要组成部分。

罗吉斯回归是广义的线性模型，就是在线性回归基础上加了一个非线性映射。逻辑回归输出的值是 0 到 1 之间的值，反应的是该点到划分线的距离，这个值不是真实的概率，但很多时候人们【假定】它是一个接近真实概率的值。

从而，根据这个概率值，作为一种【神经元】（见本节第三部分）的开关，协助多个神经元彼此之间的信号传递，无数这类信号的多层多次传递，即是神经网络的基本构造，当然，神经网络还有其他算法支撑，不全依靠逻辑回归。

（一）导入罗吉斯回归的程序

（二）导入数据

（三）查看子图

（四）拆分数据为训练集和验证集

（五）输入像素、平面、标签、变量以及模型

（六）训练模型

（七）绘制精确度曲线

（八）查看每种字体的权重的子图

接下来，我们执行上述九块内容，但是具有 14 个步骤的流程里，使用 Tensorflow 进行罗吉斯回归。首先，第一步导入 logistic 模块的程序，在 Python Shell 指令如下：

```
import tensorflow as tf
import numpy as np
try:
        from tqdm import tqdm
    except ImportError:
        def tqdm(x, *args, **kwargs):
            return x
np.random.seed(0)    # 设定随机种子
```

结果：

```
In [30]: import numpy as np

In [31]: try:
    ...:     from tqdm import tqdm
    ...: except ImportError:
    ...:     def tqdm(x, *args, **kwargs):
    ...:         return x

In [32]: np.random.seed(0)

In [33]:
```

图 7.62 导入罗吉斯回归的程序

导入数据，在 Python Shell 指令如下：

```
data = np.load('data_with_labels.npz')
 train = data['arr_0']/255.
 labels = data['arr_1']
```

结果：

```
In [33]: data = np.load('data_with_labels.npz')
    ...: train = data['arr_0']/255.
    ...: labels = data['arr_1']
Traceback (most recent call last):

  File "<ipython-input-33-a8bcccf4a9e4>", line 1, in <module>
    data = np.load('data_with_labels.npz')

  File "C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\npyio.py", line
415, in load
    fid = open(os_fspath(file), "rb")

FileNotFoundError: [Errno 2] No such file or directory: 'data_with_labels.npz'
```

图 7.63 导入数据

如图 7.63 所示，没有找到标签文件，因此我们需要再次找到工作路径，然后在工作路径下，放置数据，才可进行后续的分析工作。此处，在 Python Shell 指令如下：

```
import os
print os.getcwd()    # Python 2.X 的查询方法
print (os.getcwd()) # Python 3.X 的查询方法
```

结果：

```
In [34]: import os

In [35]: print os.getcwd()
  File "<ipython-input-35-a601e977f55c>", line 1
    print os.getcwd()
           ^
SyntaxError: invalid syntax


In [36]:

In [36]: print (os.getcwd())
C:\Users\user

In [37]: data = np.load('data_with_labels.npz')
    ...: train = data['arr_0']/255.
    ...: labels = data['arr_1']

In [38]:
```

图 7.64 确认工作路径和数据位置

此时，我们为了之后图表呈现（图 7.74），所以先查看没有图示情况下的子图，查看子图的 Python Shell 指令如下：

```
import matplotlib.pyplot as plt
plt.ion()

f, plts = plt.subplots(5, sharex=True)
```

结果：

```
In [54]: import matplotlib.pyplot as plt
    ...: plt.ion()

In [55]: import matplotlib.pyplot as plt

In [56]: plt.ion()

In [57]: f, plts = plt.subplots(5, sharex=True)
```

图 7.65 查看子图

以查看字母【A】为例，在 Python Shell 指令如下，其他字母（字体）也可照此方式查看，不再赘述：

```
f, plts = plt.subplots(5, sharex=True)
  c = 91
for i in range(5):
     plts[i].pcolor(train[c + i * 558],
                    cmap=plt.cm.gray_r)
def to_onehot(labels,nclasses = 5):
     outlabels = np.zeros((len(labels),nclasses))
     for i,l in enumerate(labels):
         outlabels[i,l] = 1
     return outlabels
onehot = to_onehot(labels)
```

结果：

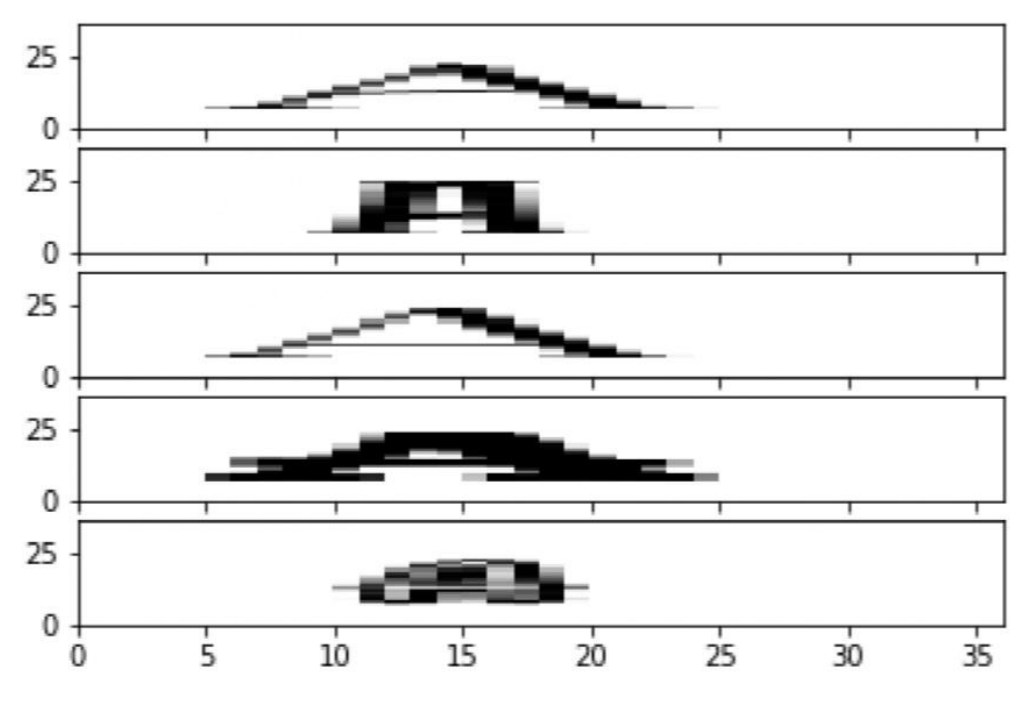


图 7.66 查看字体 A 的子图

接着，我们关闭 TF 之前的会话，做好接下来的机器分析的准备，关闭 TF 会话（不是把 TF 和 Python 关闭，而是关闭 session 准备接下来的分析）在 Python Shell 指令如下：

```
sess = tf.InteractiveSession()
## 如果您之前有开启过 tf.InteractiveSession()的话，那么现在这里需要关闭它。

sess.close()
## 系统会自动提示，需要 tf.InteractiveSession.close()的指令。因为我们更早之前定义 sess = tf.InteractiveSession()所以这里需要下指令 sess.close()来关闭，然后再此使用此处的 sess = tf.InteractiveSession()开启新的会话。
```

结果：

```
In [61]: sess = tf.InteractiveSession()
C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client
\session.py:1702: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly call
`InteractiveSession.close()` to release resources held by the other
session(s).
  warnings.warn('An interactive session is already active. This can '

In [67]: sess.close()

In [68]: sess = tf.InteractiveSession()

In [69]:
```

图 7.67 关闭 TF 之前的会话

为模型进行参数训练，首先按照机器学习的方式（见本章第一节第三部分），我们要求

拆分数据为训练集和验证集，在 Python Shell 指令如下：

```
indices = np.random.permutation(train.shape[0])
valid_cnt = int(train.shape[0] * 0.1)

test_idx, training_idx = indices[:valid_cnt],\
                         indices[valid_cnt:]
test, train = train[test_idx,:],\
              train[training_idx,:]
onehot_test, onehot_train = onehot[test_idx,:],\
                         onehot[training_idx,:])
```

结果：

```
In [60]: indices = np.random.permutation(train.shape[0])
    ...: valid_cnt = int(train.shape[0] * 0.1)
    ...: test_idx, training_idx = indices[:valid_cnt],\
    ...:                          indices[valid_cnt:]
    ...: test, train = train[test_idx,:],\
    ...:               train[training_idx,:]
    ...: onehot_test, onehot_train = onehot[test_idx,:],\
    ...:                             onehot[training_idx,:]
```

图 7.68 拆分数据为训练集和验证集

设置【输入像素、平面、标签、变量以及模型】，注意这是 InteractiveSession()函数内的诸多指令，并且实施【初始化】，也就是将函数内的诸多命令进行了运作（run）以确保在之后的工作中发挥作用。在 Python Shell 指令如下：

```
sess = tf.InteractiveSession() # 设置对话
x = tf.placeholder("float", [None, 1296])   #像素
y_ = tf.placeholder("float", [None,5])  #已知标签
W = tf.Variable(tf.zeros([1296,5])) #变量
b = tf.Variable(tf.zeros([5]))
sess.run(tf.initialize_all_variables()) #对话初始化

y = tf.nn.softmax(tf.matmul(x,W) + b)   # 定义模型
```

结果：

```
In [68]: sess = tf.InteractiveSession()

In [69]: x = tf.placeholder("float", [None, 1296])

In [70]: y_ = tf.placeholder("float", [None,5])

In [71]: W = tf.Variable(tf.zeros([1296,5]))

In [72]: b = tf.Variable(tf.zeros([5]))

In [73]: sess.run(tf.initialize_all_variables())

In [74]: y = tf.nn.softmax(tf.matmul(x,W) + b)
```

图 7.69 输入像素、平面、标签、变量以及模型

接着，着手训练模型，首先第一件事就是定义【交叉熵】，在 Python Shell 指令如下：

```
cross_entropy = tf.reduce_mean(
            tf.nn.softmax_cross_entropy_with_logits(
            logits = y + 1e-50, labels = y_))

## 未来版本的 TF 将会修改，在 tensorflow.python.ops.nn_ops 里面的
softmax_cross_entropy_with_logits 函数将被删除，成为默认状态。
```

结果：

```
In [75]: cross_entropy = tf.reduce_mean(
    ...:         tf.nn.softmax_cross_entropy_with_logits(
    ...:         logits = y + 1e-50, labels = y_))
WARNING:tensorflow:From <ipython-input-75-e85202927d74>:3:
softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is
deprecated and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.
```

图 7.70 训练模型：交叉熵

训练模型的第二步，得要告知训练方式以及准确度如何判断，这些指令如下：

```
# 告知训练方式
train_step = tf.train.GradientDescentOptimizer(
                0.02).minimize(cross_entropy)
# 定义准确度
```

```
correct_prediction = tf.equal(tf.argmax(y,1),
                     tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(
           correct_prediction, "float"))
```

表现为：

```
In [76]: train_step = tf.train.GradientDescentOptimizer(
    ...:                      0.02).minimize(cross_entropy)

In [77]: correct_prediction = tf.equal(tf.argmax(y,1),
    ...:                               tf.argmax(y_,1))
    ...: accuracy = tf.reduce_mean(tf.cast(
    ...:                correct_prediction, "float"))

In [78]:
```

图 7.71 训练模型：告知训练方式和定义准确度

第三步就是正式训练，在 Python Shell 指令如下：

```
# 训练什么
epochs = 1000
train_acc = np.zeros(epochs//10)
test_acc = np.zeros(epochs//10)
# 如何训练
for i in tqdm(range(epochs)):
    if i % 10 == 0:
        A = accuracy.eval(feed_dict={
            x: train.reshape([-1,1296]),
            y_: onehot_train})
        train_acc[i//10] = A
        A = accuracy.eval(feed_dict={
            x: test.reshape([-1,1296]),
            y_: onehot_test})
        test_acc[i//10] = A
    train_step.run(feed_dict={
        x: train.reshape([-1,1296]),
        y_: onehot_train})
# 展示结果（打印什么）
print(train_acc[-1])
print(test_acc[-1])
```

结果：

```
In [78]: epochs = 1000
    ...: train_acc = np.zeros(epochs//10)
    ...: test_acc = np.zeros(epochs//10)
    ...: for i in tqdm(range(epochs)):
    ...:     if i % 10 == 0:
    ...:         A = accuracy.eval(feed_dict={
    ...:             x: train.reshape([-1,1296]),
    ...:             y_: onehot_train})
    ...:         train_acc[i//10] = A
    ...:         A = accuracy.eval(feed_dict={
    ...:             x: test.reshape([-1,1296]),
    ...:             y_: onehot_test})
    ...:         test_acc[i//10] = A
    ...:     train_step.run(feed_dict={
    ...:         x: train.reshape([-1,1296]),
    ...:         y_: onehot_train})
    ...:
    ...: print(train_acc[-1])
    ...: print(test_acc[-1])
  7%|█          | 69/1000 [00:04<00:45, 20.29it/s]
```

图 7.72 训练模型：开始训练

图 7.73 训练模型：完成训练？

训练完成之后，我们直观看到模型预测（通过测试集的判断）的准确度，在 PythonShell 里绘制它的图表的指令如下：

```
plt.figure(figsize=(6,6))
plt.plot(train_acc,'bo')
plt.plot(test_acc,'rx')
```

结果：

图 7.74 绘制精确度曲线

最终，我们查看每种字体的权重，在 Python Shell 的指令如下：

```
f, plts = plt.subplots(5, sharex=True)
for i in range(5):
    plts[i].pcolor(W.eval()[:,i].reshape([36,36]))
```

结果：

```
In [80]: f, plts = plt.subplots(5, sharex=True)
    ...: for i in range(5):
    ...:     plts[i].pcolor(W.eval()[:,i].reshape([36,36]))
```

```
In [81]:
```

图 7.75 查看每种字体的权重子图

三、神经网络

我们先来看一组神经网络的基本构造，接着我们再考虑单隐藏层、多隐藏层以及深层神经网络的情况。通过比较，能够更为理解。

在神经网络的基本构造，包括三个部分：向量、线性变换，以及神经元。

我们在 Tensorflow 进行构建，因此需要加载模块，接着第一个要做到的是创建向量。

```
# 加载模块
import tensorflow as tf
import numpy as np
import math

# 创建向量，和 7.2.2 刚开始一样，也和 7.1.4 的练习一样。
sess = tf.InteractiveSession()
x1 = tf.Variable(tf.truncated_normal([5], mean=3, stddev=1./math.sqrt(5)))  # 长度为 5 的向量的正态随机数
x2 = tf.Variable(tf.truncated_normal([5], mean=-1, stddev=1./math.sqrt(5))) #以不同中心点 3,-1,0 截断，避免极端。
x3 = tf.Variable(tf.truncated_normal([5], mean=0, stddev=1./math.sqrt(5)))
sess.run(tf.global_variables_initializer())
```

结果：

```
In [1]: import tensorflow as tf

In [2]: import numpy as np

In [3]: import math

In [4]: sess = tf.InteractiveSession()

In [5]: x1 = tf.Variable(tf.truncated_normal([5], mean=3, stddev=1./math.sqrt(5)))
   ...: x2 = tf.Variable(tf.truncated_normal([5], mean=-1, stddev=1./math.sqrt(5)))
   ...: x3 = tf.Variable(tf.truncated_normal([5], mean=0, stddev=1./math.sqrt(5)))
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\pytho
\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.

In [6]: sess.run(tf.global_variables_initializer())
```

图 7.75 加载模块和创建向量

我们有了向量之后，我们得要制定【线性变换】方式，其代码如下：

```
sqx2 = x2 * x2       #采用平方变换
print(x2.eval()) #平方会使数值变成正数，极端数值放大。
print(sqx2.eval())
logx1 = tf.log(x1)  #采用 Log 函数
print(x1.eval())      #对数会使较小数值的差别凸显。
print(logx1.eval())
sigx3 = tf.sigmoid(x3)  #采用 sigmoid 函数
print(x3.eval())     #任何正负数值近似 1 和 0
print(sigx3.eval())
## Sigmoid 函数，会使接近 0 的数值变成接近 1/2
```

结果：

```
In [7]: sqx2 = x2 * x2

In [8]: print(x2.eval())
[-0.83629084 -1.1042433  -0.83839905 -0.68164134 -1.3791976 ]

In [9]: print(sqx2.eval())
[0.69938236 1.2193532  0.702913   0.46463493 1.902186  ]

In [10]: logx1 = tf.log(x1)

In [11]: print(x1.eval())
[2.779665  2.7063272 3.2050529 3.0966372 3.2410536]

In [12]: print(logx1.eval())
[1.0223304 0.9955925 1.1647286 1.1303167 1.1758984]

In [13]: sigx3 = tf.sigmoid(x3)

In [14]: print(x3.eval())
[-0.2412223   0.16060764 -0.34726354 -0.12940934 -0.5273337 ]

In [15]: print(sigx3.eval())
[0.43998516 0.5400658  0.41404614 0.46769273 0.371139  ]
```

图 7.76 线性变换

就神经网络而言，完成【神经元】的构造，才算完成神经网络的基础单元，也才刚刚开

启了神经网络的分析或者后续的深度学习的开始。

```
w1 = tf.constant(0.1)   #创建常量
w2 = tf.constant(0.2)
sess.run(tf.global_variables_initializer())#初始化变量
n1 = tf.sigmoid(w1*x1 + w2*x2) # 激活 sigmoid 函数
print((w1*x1).eval()) #
print((w2*x2).eval()) #
print(n1.eval()) #多个输入之后的线性转换
# 组成神经网络：一个神经元的输入成为下一层神经元的输入，通过权重以及非线性函数（例如 Sigmoid 等）可有效计算出最终模型的一组新的特征。
```

结果：

```
In [16]: w1 = tf.constant(0.1)
    ...: w2 = tf.constant(0.2)

In [17]: sess.run(tf.global_variables_initializer())

In [18]: n1 = tf.sigmoid(w1*x1 + w2*x2)

In [19]: print((w1*x1).eval())
[0.30683988 0.31245348 0.30205142 0.27959928 0.26084337]

In [20]: print((w2*x2).eval())
[-0.2891154  -0.12513871 -0.16808121 -0.07866509 -0.19285798]

In [21]: print(n1.eval())
[0.504431   0.54669225 0.53344256 0.5500652  0.51698977]

In [22]:
```

图 7.77 构造神经元

接下来，我们实际上进行本节第二部分的同样工作，只不过现在不是罗吉斯回归，而是采用了神经网络的方法。主要分为四大流程和 11 项任务：

（一）数据：查看数据、转换格式、拆分训练集与测试集；

（二）设定：隐藏层、输出层、模型；

（三）训练：交叉熵、训练模型；

（四）结果：精确度、混淆矩阵、权重。

以下，第一步，我们进行数据处理和数据规整，其代码如下：

```
# 导入模块
    import tensorflow as tf
    import numpy as np
    import math
    try:
         from tqdm import tqdm
      except ImportError:
         def tqdm(x, *args, **kwargs):
             return x
# 导入数据、定义标签
```

```
    data = np.load('data_with_labels.npz')
    train = data['arr_0']/255.
    labels = data['arr_1']

# 查看数据
    print(train[0])
    print(labels[0])
```

接着，我们转换格式，其代码如下：

```
# 导入绘图程序
    import matplotlib.pyplot as plt
    plt.ion()

# 转换标签
    def to_onehot(labels,nclasses = 5):
         outlabels = np.zeros((len(labels),nclasses))
         for i,l in enumerate(labels):
             outlabels[i,l] = 1
         return outlabels
    onehot = to_onehot(labels)
```

这一流程的最后一步，我们遵循机器学习的方式，拆分数据，其代码如下：

```
    # 定义如何拆分数据集为训练集与测试集
    indices = np.random.permutation(train.shape[0])
    valid_cnt = int(train.shape[0] * 0.1)
    test_idx, training_idx = indices[:valid_cnt], indices[valid_cnt:]
    test, train = train[test_idx,:], train[training_idx,:]
    onehot_test, onehot_train = onehot[test_idx,:], onehot[training_idx,:]
```

第二步，设定神经网络模型，其代码如下：

```
# 启动对话
sess = tf.InteractiveSession()

# 输入浮点数和标签
x = tf.placeholder("float", [None, 1296])
y_ = tf.placeholder("float", [None,5])

# 构建隐藏层
num_hidden = 128    #设定 128 个数。
W1                    =                   tf.Variable(tf.truncated_normal([1296,
num_hidden],stddev=1./math.sqrt(1296)))
b1 = tf.Variable(tf.constant(0.1,shape=[num_hidden]))
h1 = tf.sigmoid(tf.matmul(x,W1) + b1)

# 构建输出层
W2 = tf.Variable(tf.truncated_normal([num_hidden, 5], stddev=1./math.sqrt(5)))
b2 = tf.Variable(tf.constant(0.1,shape=[5]))
```

```
# 初始化 （运行会话）
sess.run(tf.global_variables_initializer())
```

上面这段代码是 Session 会话，但这一步还未走完，最重要的是下面这句代码：

```
    # 定义模型
    y = tf.nn.softmax(tf.matmul(h1,W2) + b2)
```

第三步，我们自然是开始训练模型，但其中，要先定义【交叉熵】以及【如何进行训练】，其代码如下，这些做法与前面第二部分如出一撤：

```
# 定义【交叉熵】
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = y
+ 1e-50, labels = y_))

# 定义如何进行训练
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

# 定义【精确度】
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

好了，我们终于开始训练了，其代码如下：

```
# 定义数量、训练集以及测试集
epochs = 5000
train_acc = np.zeros(epochs//10)
test_acc = np.zeros(epochs//10)

for i in tqdm(range(epochs), ascii=True):
     if i % 10 == 0:  # 返回数据情况以及精确度
     A = accuracy.eval(feed_dict={x: train.reshape([-1,1296]), y_:onehot_train})
train_acc[i//10] = A
     A = accuracy.eval(feed_dict={x: test.reshape([-1,1296]), y_: onehot_test})
test_acc[i//10] = A
     train_step.run(feed_dict={x: train.reshape([-1,1296]), y_: onehot_train})
```

结果：

```
In [43]: for i in tqdm(range(epochs), ascii=True):
    ...:     if i % 10 == 0:
    ...:         A = accuracy.eval(feed_dict={x: train.reshape([-1,1296]), y_:
onehot_train})
    ...:         train_acc[i//10] = A
    ...:         A = accuracy.eval(feed_dict={x: test.reshape([-1,1296]), y_:
onehot_test})
    ...:         test_acc[i//10] = A
    ...:     train_step.run(feed_dict={x: train.reshape([-1,1296]), y_:
onehot_train})

  0%|          | 0/5000 [00:00<?, ?it/s]
  0%|          | 1/5000 [00:00<17:14,  4.83it/s]
  0%|          | 2/5000 [00:00<15:32,  5.36it/s]
  0%|          | 3/5000 [00:00<14:13,  5.85it/s]
  0%|          | 4/5000 [00:00<12:45,  6.53it/s]
  0%|          | 5/5000 [00:00<11:53,  7.00it/s]
  0%|          | 6/5000 [00:00<11:10,  7.45it/s]
  0%|          | 7/5000 [00:00<10:41,  7.78it/s]
  0%|          | 9/5000 [00:01<09:54,  8.39it/s]
  0%|          | 11/5000 [00:01<09:41,  8.58it/s]
```

图 9.78 开始训练

```
 40%|###9      | 1988/5000 [02:57<03:03, 16.46it/s]
 40%|###9      | 1990/5000 [02:57<03:03, 16.40it/s]
 40%|###9      | 1992/5000 [02:57<03:14, 15.48it/s]
 40%|###9      | 1994/5000 [02:57<03:09, 15.86it/s]
 40%|###9      | 1996/5000 [02:57<03:04, 16.25it/s]
 40%|###9      | 1998/5000 [02:57<03:01, 16.57it/s]
 40%|####      | 2000/5000 [02:57<02:58, 16.85it/s]
 40%|####      | 2002/5000 [02:58<03:07, 15.99it/s]
 40%|####      | 2004/5000 [02:58<03:02, 16.39it/s]
 40%|####      | 2006/5000 [02:58<02:59, 16.68it/s]
 40%|####      | 2008/5000 [02:58<02:57, 16.88it/s]
 40%|####      | 2010/5000 [02:58<02:59, 16.69it/s]
 40%|####      | 2012/5000 [02:58<03:11, 15.59it/s]
 40%|####      | 2014/5000 [02:58<03:06, 16.05it/s]
 40%|####      | 2016/5000 [02:58<03:00, 16.51it/s]
 40%|####      | 2018/5000 [02:58<02:56, 16.85it/s]
 40%|####      | 2020/5000 [02:59<02:55, 16.97it/s]
 40%|####      | 2022/5000 [02:59<03:04, 16.10it/s]
 40%|####      | 2024/5000 [02:59<03:01, 16.39it/s]
 41%|####      | 2026/5000 [02:59<02:57, 16.72it/s]
 41%|####      | 2028/5000 [02:59<02:54, 17.04it/s]
 41%|####      | 2030/5000 [02:59<02:52, 17.19it/s]
```

```
 99%|#########9| 4969/5000 [07:37<00:04,  7.50it/s]
 99%|#########9| 4970/5000 [07:37<00:03,  7.82it/s]
 99%|#########9| 4971/5000 [07:37<00:04,  7.12it/s]
 99%|#########9| 4972/5000 [07:38<00:03,  7.44it/s]
 99%|#########9| 4973/5000 [07:38<00:03,  7.55it/s]
100%|#########9| 4975/5000 [07:38<00:03,  8.24it/s]
100%|#########9| 4976/5000 [07:38<00:02,  8.52it/s]
100%|#########9| 4977/5000 [07:38<00:02,  8.44it/s]
100%|#########9| 4979/5000 [07:38<00:02,  9.10it/s]
100%|#########9| 4981/5000 [07:39<00:02,  9.13it/s]
100%|#########9| 4983/5000 [07:39<00:01,  9.72it/s]
100%|#########9| 4985/5000 [07:39<00:01, 10.51it/s]
100%|#########9| 4987/5000 [07:39<00:01, 11.40it/s]
100%|#########9| 4989/5000 [07:39<00:00, 12.16it/s]
100%|#########9| 4991/5000 [07:39<00:00, 11.72it/s]
100%|#########9| 4993/5000 [07:39<00:00, 12.50it/s]
100%|#########9| 4995/5000 [07:40<00:00, 13.12it/s]
100%|#########9| 4997/5000 [07:40<00:00, 13.26it/s]
100%|#########9| 4999/5000 [07:40<00:00, 13.12it/s]
100%|##########| 5000/5000 [07:40<00:00, 10.86it/s]

In [44]:
```

图 7.79 训练结束

第四步，看图说事，让 Python 为我们描绘精确度的曲线吧！

```
plt.figure(figsize=(6,6))
plt.plot(train_acc,'bo')
plt.plot(test_acc,'rx')
```

结果：

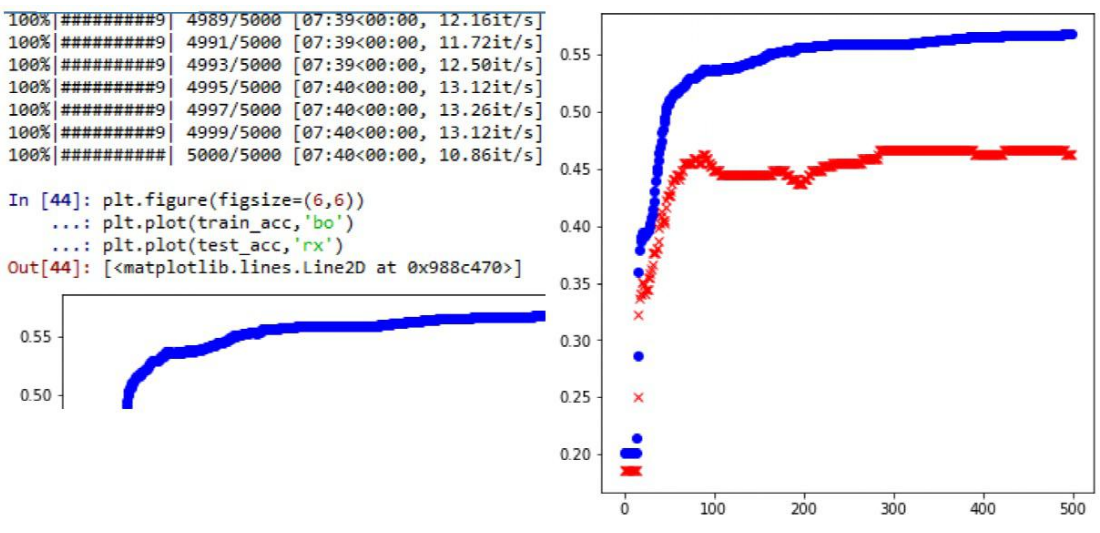


图 7.80 精确度的曲线

除了精确度，混淆矩阵和权重，是机器学习最为看重的另外两个“成果”，所以我们得要查看最终测试的混淆矩阵，其代码如下：

```
pred = np.argmax(y.eval(feed_dict={x: test.reshape([-1,1296]), y_: onehot_test}),
axis = 1)
conf = np.zeros([5,5])
for p,t in zip(pred,np.argmax(onehot_test,axis=1)):
    conf[t,p] += 1

plt.matshow(conf)
plt.colorbar()
```

结果：

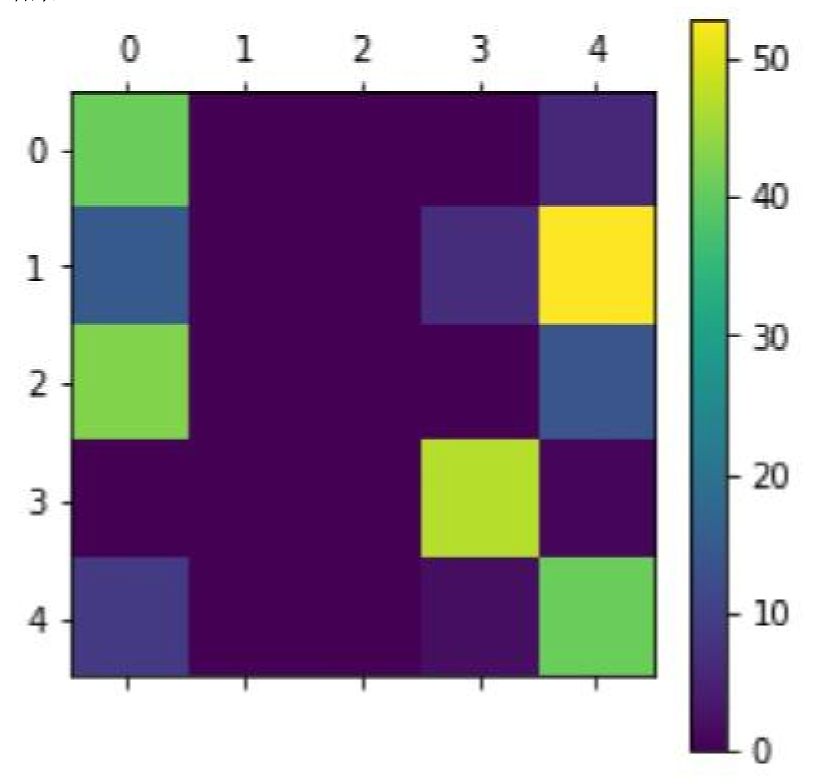


图 7.81 最终测试的混淆矩阵

最后，查看权重，其代码如下：

```
plt.figure(figsize=(6,6))
 f, plts = plt.subplots(4,8, sharex=True)
 for i in range(32):
    plts[i//8, i%8].pcolormesh(W1.eval()[:,i].reshape([36,36]))
```

结果：

```
In [50]: plt.figure(figsize=(6,6))
    ...: f, plts = plt.subplots(4,8, sharex=True)
    ...: for i in range(32):
    ...:     plts[i//8, i%8].pcolormesh(W1.eval()[:,i].reshape([36,36]))
<Figure size 432x432 with 0 Axes>
```

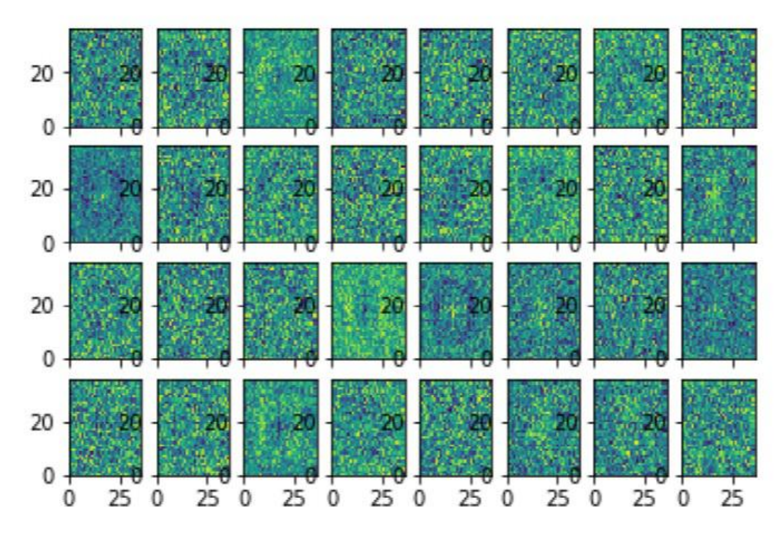


图 7.82 查看权重

以上，神经网络处理像素的数据集，又称为单隐层神经网络，因为只有一个隐藏层，与之对应的是【二个以及两个以上】的隐藏层的情况，称为多隐藏层，在下面第四部分介绍。

四、多隐藏层神经网络

本小节作为一个相对完整的工作流程，能够主要反应神经网络，特别是以多隐藏层神经网络的方式，进行机器学习之后的过程。

与前面第三部分相比，在本小节的多隐藏层神经网络，主要有两处不同：

（一）隐藏层从一个变成两个（多层的话，可以不止两个）

（二）把模型参数，即：权重，视为一种数据，进行存储和取用。

与前面一部分类似，我们也把任务归纳为四个流程，其中略有不同。

（一）数据：读取数据、转换数据、拆分数据；

（二）设定：隐藏层（有两个）、输出层、模型；

（三）训练：交叉熵、训练法、预测法、精确度、训练模型；

（四）结果：精确度、混淆矩阵、权重（增加权重存取）。

第一步数据处理的部分，我们需要读取数据、转换数据和拆分数据。读取数据的代码和第三部分的如出一撤，如下：

```
import tensorflow as tf
import numpy as np
import math
try:
     from tqdm import tqdm
  except ImportError:
     def tqdm(x, *args, **kwargs):
         return x
data = np.load('data_with_labels.npz')
train = data['arr_0']/255.
labels = data['arr_1']

print(train[0])
print(labels[0])
```

转换数据的指令如下：

```
import matplotlib.pyplot as plt
plt.ion()
def to_onehot(labels,nclasses = 5):
    outlabels = np.zeros((len(labels),nclasses))
    for i,l in enumerate(labels):
        outlabels[i,l] = 1
    return outlabels
onehot = to_onehot(labels)
```

拆分数据的指令如下：

```
indices = np.random.permutation(train.shape[0])
valid_cnt = int(train.shape[0] * 0.1)
test_idx,training_idx=indices[:valid_cnt],indices[valid_cnt:]
test, train = train[test_idx,:], train[training_idx,:]
onehot_test, onehot_train = onehot[test_idx,:], onehot[training_idx,:]
```

第二步，设定神经网络模型，这里我们做了两个隐藏层，其代码如下：

```
# 输入数据
sess = tf.InteractiveSession()
x = tf.placeholder("float", [None, 1296])
y_ = tf.placeholder("float", [None,5])

#第一隐藏层
num_hidden1 = 128
W1 = tf.Variable(tf.truncated_normal([1296,num_hidden1],
stddev=1./math.sqrt(1296)))
b1 = tf.Variable(tf.constant(0.1,shape=[num_hidden1]))
h1 = tf.sigmoid(tf.matmul(x,W1) + b1)
```

```
#第二隐藏层
num_hidden2 = 32
W2 = tf.Variable(tf.truncated_normal([num_hidden1,
num_hidden2],stddev=2./math.sqrt(num_hidden1)))
b2 = tf.Variable(tf.constant(0.2,shape=[num_hidden2]))
h2 = tf.sigmoid(tf.matmul(h1,W2) + b2)

#输出层
W3 = tf.Variable(tf.truncated_normal([num_hidden2, 5], stddev=1./math.sqrt(5)))
b3 = tf.Variable(tf.constant(0.1,shape=[5]))
sess.run(tf.initialize_all_variables())

#定义模型
y = tf.nn.softmax(tf.matmul(h2,W3) + b3)
```

第三步是训练模型，以下代码，与第三部分的都一样。

```
#交叉熵
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = y
+ 1e-50, labels = y_))
#训练法
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
#精确度
correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y_,1))
accuracy=tf.reduce_mean(tf.cast(correct_prediction, "float"))

# 训练数据
epochs = 25000
train_acc = np.zeros(epochs//10)
test_acc = np.zeros(epochs//10)
for i in tqdm(range(epochs), ascii=True):
         if i % 10 == 0:
             A = accuracy.eval(feed_dict={x: train.reshape([-1,1296]), y_:
onehot_train})
             train_acc[i//10] = A
            A = accuracy.eval(feed_dict={x: test.reshape([-1,1296]), y_:
onehot_test})
             test_acc[i//10] = A
         train_step.run(feed_dict={x: train.reshape([-1,1296]), y_:
onehot_train})
```

因为多隐藏层神经网络和单隐藏层神经网络，在模型上，还是不太一样，所以我们开始训练数据求取模型之后，截图如下：

```
In [33]: epochs = 25000
    ...: train_acc = np.zeros(epochs//10)
    ...: test_acc = np.zeros(epochs//10)
    ...: for i in tqdm(range(epochs), ascii=True):
    ...:     if i % 10 == 0:
    ...:         A = accuracy.eval(feed_dict={x: train.reshape([-1,1296]), y_:
onehot_train})
    ...:         train_acc[i//10] = A
    ...:         A = accuracy.eval(feed_dict={x: test.reshape([-1,1296]), y_:
onehot_test})
    ...:         test_acc[i//10] = A
    ...:     train_step.run(feed_dict={x: train.reshape([-1,1296]), y_:
onehot_train})
  6%|5          | 1495/25000 [01:38<24:09, 16.21it/s]
```

图 7.83 训练数据

图 7.83 和图 7.78 是差不多的，实际上就是运算时间较长，但是因为我们的数据量很小，而且隐藏层也只是多增加一层，所以个人感觉不到等待了更长的运算时间。

第四步要查看结果。其代码如下：

```
# 查看精确度曲线、混淆矩阵和权重
plt.figure(figsize=(6,6))
     plt.plot(train_acc,'bo')
     plt.plot(test_acc,'rx')

pred = np.argmax(y.eval(feed_dict={x: test.reshape([-1,1296]), y_: onehot_test}),
axis = 1)
     conf = np.zeros([5,5])
 for p,t in zip(pred,np.argmax(onehot_test,axis=1)):
         conf[t,p] += 1
plt.matshow(conf)
plt.colorbar()
f, plts = plt.subplots(4,8, sharex=True)
     for i in range(32):
         plts[i//8,i%8].matshow(W1.eval()[:,i].reshape([36,36]))
```

我们增加了一段代码：

```
检查权重、存储，恢复
plt.matshow(W3.eval())
     plt.colorbar()

saver = tf.train.Saver()
     saver.save(sess, "mpl.ckpt")

saver.restore(sess, "mlp.ckpt")
```

结果：

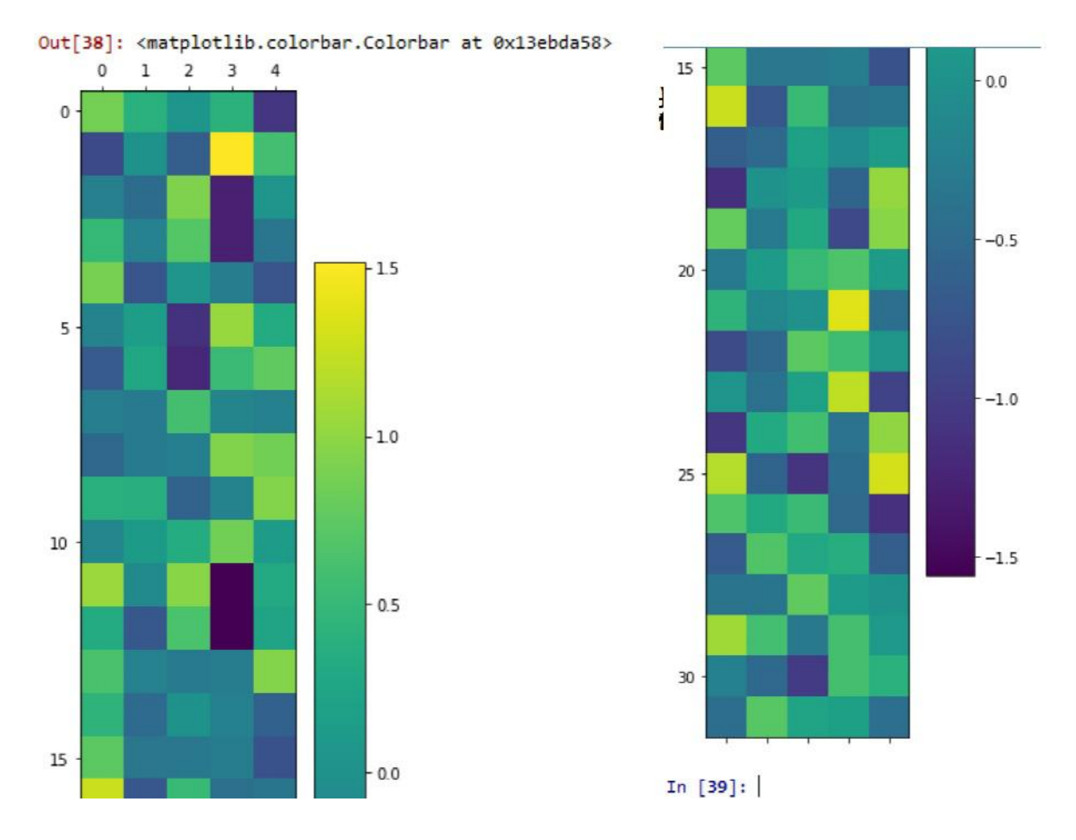


图 7.84 检查权重

过去我们讨论数据处理（第二章）时，多半以数据框进行讨论，包括这种版本的数据框的存储与取用，即便讨论列表、矩阵、字典时，也多半是作为对象（object）予以存储，此处，我们涉及到具有像素的图片经过数据处理之后进行模型运算，其模型所训练出来的权重，把权重视为一个重要的数据，予以存储和取用。

第三节 深度学习

一、深层神经网络

在这一节开始真正进入深度学习了，本小节的深层神经网络，运用到前面第二节中的罗吉斯回归、神经元、多隐藏层神经网络，但是还有添加单线训练以及密集神经网络分类器等部分。深层神经网络只是从机器学习迈向深度学习的一个开始，也是从多隐藏层神经网络迈向深度学习的一个初端，往后在此基础上，根据不同问题、数据和求解目的产生多个不同的深度学习。

为求方便理解，我们拆解整个流程，而不采取.py 文件的方式，在 Python Shell 上逐条代码输入，以把结果进行截图的方式，列出八段内容，如下演示。

（一）准备环境

（二）导入数据

（三）拆分数据

（四）转换数据

（五）罗吉斯回归

（六）单线模型

（七）密集神经网

（八）精确度计算

```
import tensorflow.contrib.learn as learn
from tensorflow.contrib.learn.python.learn.estimators import estimator
import numpy as np
import math
import matplotlib.pyplot as plt
plt.ion()
import sklearn #机器学习的库。
from sklearn import metrics
```

结果：

```
In [38]: import tensorflow.contrib.learn as learn
    ...: from tensorflow.contrib.learn.python.learn.estimators import
estimator
    ...: import numpy as np
    ...: import math
    ...: import matplotlib.pyplot as plt
    ...: plt.ion()
    ...: import sklearn
    ...: from sklearn import metrics
```

图 7.85 准备环境

```
np.random.seed(42) # 建立随机种子
data = np.load('data_with_labels.npz')
train = data['arr_0']/255.
labels = data['arr_1']
```

结果：

```
In [39]: np.random.seed(42)
    ...: data = np.load('data_with_labels.npz')
    ...: train = data['arr_0']/255.
    ...: labels = data['arr_1']
```

图 7.86 导入数据

```
indices = np.random.permutation(train.shape[0])
valid_cnt = int(train.shape[0] * 0.1)
test_idx, training_idx = indices[:valid_cnt], indices[valid_cnt:]
test, train = train[test_idx,:], train[training_idx,:]
test_labels, train_labels = labels[test_idx], labels[training_idx]
```

结果：

```
In [40]: indices = np.random.permutation(train.shape[0])
    ...: valid_cnt = int(train.shape[0] * 0.1)
    ...: test_idx, training_idx = indices[:valid_cnt],
indices[valid_cnt:]
    ...: test, train = train[test_idx,:], train[training_idx,:]
    ...: test_labels, train_labels = labels[test_idx],
labels[training_idx]

In [41]: train = np.array(train,dtype=np.float32)
    ...: test = np.array(test,dtype=np.float32)
    ...: train_labels = np.array(train_labels,dtype=np.int32)
    ...: test_labels = np.array(test_labels,dtype=np.int32)
```

图 7.87 拆分数据

```
train = np.array(train,dtype=np.float32) # 浮点数和数组格式
test = np.array(test,dtype=np.float32)
train_labels = np.array(train_labels,dtype=np.int32)
test_labels = np.array(test_labels,dtype=np.int32)

feature_columns                                                        =
learn.infer_real_valued_columns_from_input(train.reshape([-1,36*36]))
## 转换特征为机器学习的格式
```

结果：

```
In [42]: feature_columns =
learn.infer_real_valued_columns_from_input(train.reshape([-1,36*36]))
WARNING:tensorflow:From <ipython-input-42-ad8f2e37dba6>:1:
infer_real_valued_columns_from_input (from
tensorflow.contrib.learn.python.learn.estimators.estimator) is
deprecated and will be removed in a future version.
Instructions for updating:
Please specify feature columns explicitly.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\contrib\learn\python\learn\estimators\estimator.py:143:
setup_train_data_feeder (from
tensorflow.contrib.learn.python.learn.learn_io.data_feeder) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use tensorflow/transform or tf.data.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\contrib\learn\python\learn\learn_io\data_feeder.py:96:
extract_dask_data (from
tensorflow.contrib.learn.python.learn.learn_io.dask_io) is deprecated
and will be removed in a future version.
Instructions for updating:
Please feed input to tf.data to support dask.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\contrib\learn\python\learn\learn_io\data_feeder.py:100:
and will be removed in a future version.
Instructions for updating:
Please access pandas data directly.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\contrib\learn\python\learn\learn_io\data_feeder.py:159:
DataFeeder.__init__ (from
tensorflow.contrib.learn.python.learn.learn_io.data_feeder) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use tensorflow/transform or tf.data.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\contrib\learn\python\learn\learn_io\data_feeder.py:340:
check_array (from
tensorflow.contrib.learn.python.learn.learn_io.data_feeder) is
deprecated and will be removed in a future version.
Instructions for updating:
Please convert numpy dtypes explicitly.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\contrib\learn\python\learn\estimators\estimator.py:183:
infer_real_valued_columns_from_input_fn (from
tensorflow.contrib.learn.python.learn.estimators.estimator) is
deprecated and will be removed in a future version.
Instructions for updating:
Please specify feature columns explicitly.
```

图 7.88 转换数据

```
classifier = estimator.SKCompat(learn.LinearClassifier(
 feature_columns = feature_columns, n_classes=5))
```

结果：

```
In [44]: classifier =
estimator.SKCompat(learn.LinearClassifier(feature_columns =
feature_columns, n_classes=5))
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users
\user\AppData\Local\Temp\tmprngvrlnv
INFO:tensorflow:Using config: {'_task_type': None, '_task_id': 0,
'_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec
object at 0x000000000E3E0470>, '_master': '', '_num_ps_replicas': 0,
'_num_worker_replicas': 0, '_environment': 'local', '_is_chief': True,
'_evaluation_master': '', '_train_distribute': None,
'_eval_distribute': None, '_device_fn': None, '_tf_config': gpu_options
{
  per_process_gpu_memory_fraction: 1.0
}
, '_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_secs': 600, '_log_step_count_steps': 100,
'_protocol': None, '_session_config': None, '_save_checkpoints_steps':
None, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours':
10000, '_model_dir': 'C:\\Users\\user\\AppData\\Local\\Temp\
\tmprngvrlnv'}
```

图 7.89 罗吉斯回归

```
# 单线训练（One line training）
classifier.fit(train.reshape([-1,36*36]),        train_labels,        steps=1024,
batch_size=32)
## 步骤数是所有批次的数。
## num_epochs = steps*batch_size/len(train)
```

```
# 查看精确度
test_probs = classifier.predict(test.reshape([-1,36*36]))
sklearn.metrics.accuracy_score(test_labels, test_probs['classes'])
## 使用 sklearn 的精确度的函数
```

结果：

```
In [45]: classifier.fit(train.reshape([-1,36*36]), train_labels,
steps=1024, batch_size=32)
    ...: test_probs = classifier.predict(test.reshape([-1,36*36]))
    ...: sklearn.metrics.accuracy_score(test_labels,
test_probs['classes'])
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\contrib\learn\python\learn\learn_io\data_feeder.py:98:
extract_dask_labels (from
tensorflow.contrib.learn.python.learn.learn_io.dask_io) is deprecated
and will be removed in a future version.
Instructions for updating:
Please feed input to tf.data to support dask.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\contrib\learn\python\learn\learn_io\data_feeder.py:102:
extract_pandas_labels (from
tensorflow.contrib.learn.python.learn.learn_io.pandas_io) is deprecated
and will be removed in a future version.
Instructions for updating:
Please access pandas data directly.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Colocations handled automatically by placer.
```

```
INFO:tensorflow:global_step/sec: 289.838
INFO:tensorflow:loss = 0.6521285, step = 601 (0.344 sec)
INFO:tensorflow:global_step/sec: 401.584
INFO:tensorflow:loss = 0.6777919, step = 701 (0.249 sec)
INFO:tensorflow:global_step/sec: 418.386
INFO:tensorflow:loss = 0.8701743, step = 801 (0.240 sec)
INFO:tensorflow:global_step/sec: 371.726
INFO:tensorflow:loss = 0.5640015, step = 901 (0.269 sec)
INFO:tensorflow:global_step/sec: 403.203
INFO:tensorflow:loss = 0.7526468, step = 1001 (0.247 sec)
INFO:tensorflow:Saving checkpoints for 1024 into C:\Users\user\AppData
\Local\Temp\tmprngvrlnv\model.ckpt.
INFO:tensorflow:Loss for final step: 0.48464337.
INFO:tensorflow:Graph was finalized.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages
\tensorflow\python\training\saver.py:1266: checkpoint_exists (from
tensorflow.python.training.checkpoint_management) is deprecated and
will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from C:\Users\user\AppData\Local
\Temp\tmprngvrlnv\model.ckpt-1024
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Out[45]: 0.7526881720430108
```

图 7.90 单线模型

```
# 密集神经网
classifier = estimator.SKCompat(learn.DNNClassifier( feature_columns = feature_columns, hidden_units=[10,5], n_classes=5, optimizer='Adam'))

# 召回相同的训练集数据
classifier.fit(train.reshape([-1,36*36]), train_labels, steps=1024, batch_size=32)
```

结果：

```
In [46]: classifier =
estimator.SKCompat(learn.DNNClassifier(feature_columns =
feature_columns, hidden_units=[10,5], n_classes=5, optimizer='Adam'))
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users
\user\AppData\Local\Temp\tmpqcu_0bjp
INFO:tensorflow:Using config: {'_task_type': None, '_task_id': 0,
'_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec
object at 0x00000000140C1940>, '_master': '', '_num_ps_replicas': 0,
'_num_worker_replicas': 0, '_environment': 'local', '_is_chief': True,
'_evaluation_master': '', '_train_distribute': None,
'_eval_distribute': None, '_device_fn': None, '_tf_config': gpu_options
{
  per_process_gpu_memory_fraction: 1.0
}
, '_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_secs': 600, '_log_step_count_steps': 100,
'_protocol': None, '_session_config': None, '_save_checkpoints_steps':
None, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours':
10000, '_model_dir': 'C:\\Users\\user\\AppData\\Local\\Temp\
\tmpqcu_0bjp'}
```

```
In [47]: classifier.fit(train.reshape([-1,36*36]), train_labels,
steps=1024, batch_size=32)
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into C:\Users\user\AppData
\Local\Temp\tmpqcu_0bjp\model.ckpt.
INFO:tensorflow:loss = 1.642602, step = 1
INFO:tensorflow:global_step/sec: 210.958
INFO:tensorflow:loss = 1.6046603, step = 101 (0.478 sec)
INFO:tensorflow:global_step/sec: 276.227
INFO:tensorflow:loss = 1.6169181, step = 201 (0.362 sec)
INFO:tensorflow:global_step/sec: 275.466
INFO:tensorflow:loss = 1.5982757, step = 301 (0.363 sec)
INFO:tensorflow:global_step/sec: 273.208
INFO:tensorflow:loss = 1.6156082, step = 401 (0.366 sec)
INFO:tensorflow:global_step/sec: 286.516
INFO:tensorflow:loss = 1.6302893, step = 501 (0.349 sec)
INFO:tensorflow:global_step/sec: 308.624
INFO:tensorflow:loss = 1.6059089, step = 601 (0.324 sec)
INFO:tensorflow:global_step/sec: 312.482
INFO:tensorflow:loss = 1.5965108, step = 701 (0.319 sec)
INFO:tensorflow:global_step/sec: 306.731
INFO:tensorflow:loss = 1.6279207, step = 801 (0.327 sec)
```

图 7.91 密集神经网

```
# 简化精确度
test_probs = classifier.predict(test.reshape([-1,36*36]))
sklearn.metrics.accuracy_score(test_labels, test_probs['classes'])

# 容易产生混淆
train_probs = classifier.predict(train.reshape([-1,36*36]))
conf = metrics.confusion_matrix(train_labels, train_probs['classes'])
print(conf)
```

结果：

```
In [48]: test_probs = classifier.predict(test.reshape([-1,36*36]))
    ...: sklearn.metrics.accuracy_score(test_labels,
test_probs['classes'])
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\user\AppData\Local
\Temp\tmpqcu_0bjp\model.ckpt-1024
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Out[48]: 0.1863799283154122

In [49]: train_probs = classifier.predict(train.reshape([-1,36*36]))
    ...: conf = metrics.confusion_matrix(train_labels,
train_probs['classes'])
    ...: print(conf)
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\user\AppData\Local
\Temp\tmpqcu_0bjp\model.ckpt-1024
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
[[  0 500   0   0   0]
 [  0 506   0   0   0]
 [  0 499   0   0   0]
 [  0 505   0   0   0]
 [  0 501   0   0   0]]
```

图 7.92 精确度计算

二、卷积神经网络

卷积神经网络也是层级，但是【层】的功能和形式有所变化，大致上分为五个重要部分：

（一）数据输入层（Input layer）

（二）卷积计算层（CONV layer）

（三）激励层，多以 ReLU 激励层（ReLU layer）

（四）池化层（Pooling layer）

（五）输出层（FC layer）

其中，输入层和输出层，比较容易理解，但是卷积计算层、激励层以及池化层，是相对其它深度学习比较特殊的层面。

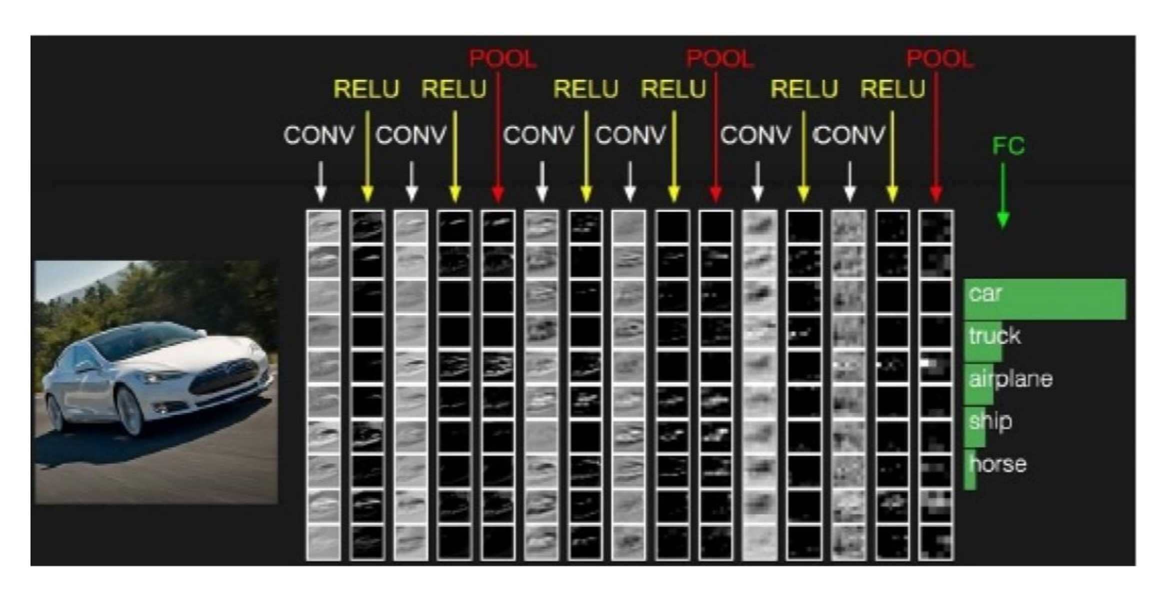


图 7.93 卷积神经网络的示意图

我们开始编写卷积神经网络的代码，其目的是帮助了解本节内容。

```
#导入程序和读取数据
import tensorflow as tf
import math
import numpy as np
sess = tf.InteractiveSession()
image = np.random.randint(10,size=[1,10,10]) + np.eye(10)*10
```

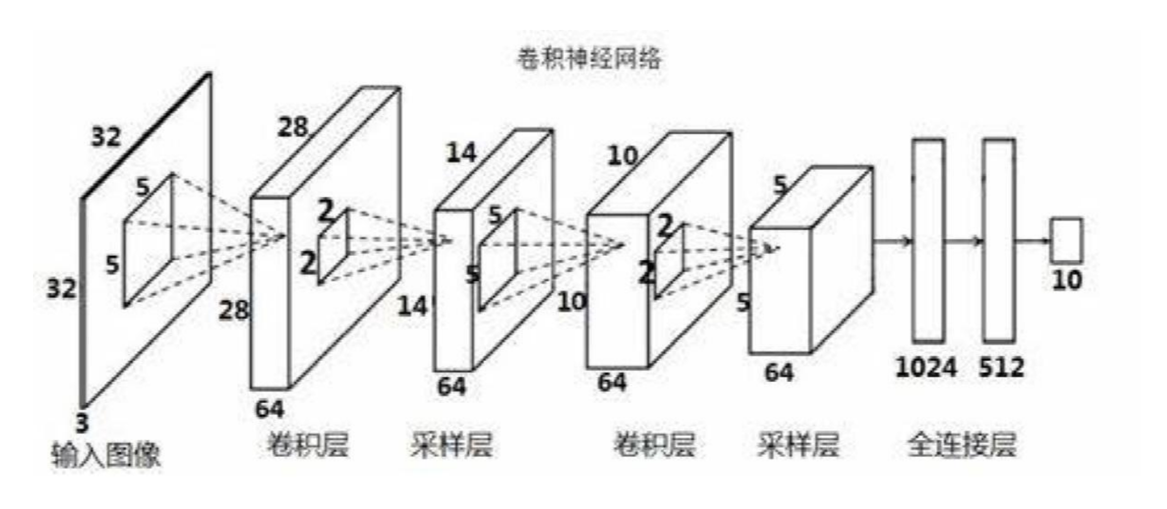


图 7.94 卷积神经网络的层级结构

数据输入层有三种图像数据处理的方式：

（一）去均值：把输入数据各个维度都中心化到 0

（二）归一化：幅度归一化到同样的范围

（三）PCA/白化：用 PCA 降维；白化是对数据每个特征轴上的幅度归一化 。

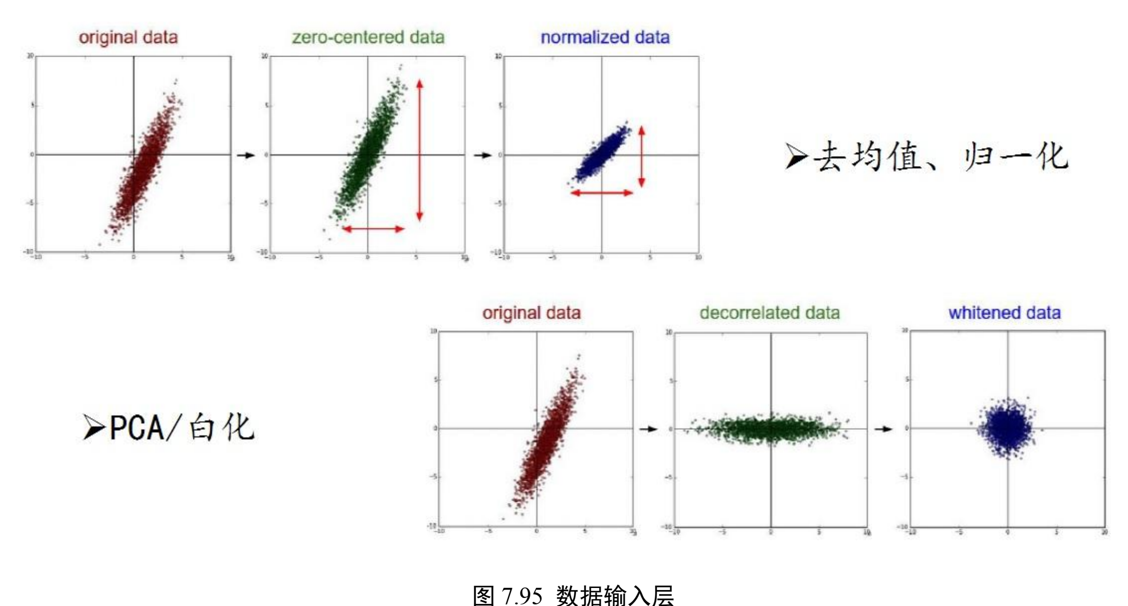


图 7.95 数据输入层

数据输入层，我们在前面第二节和本节第一部分都有类似处理经验；以下为了便于理解，我们开始编写一些卷积神经网络的基础内容，最后在图 7.100 那里可以看到结果。

```
#转换数据
x = tf.placeholder(“float”, [None, 10, 10])
##占位符函数。
##None 是用于批处理（batch processing）而 10,10 是像素。

x_im = tf.reshape(x, [-1,10,10,1])
## 设定-1 是为了保持（与 1,10,10）相同大小，而 1 是输入通道 channel 的个数（黑白灰），如果是三色图像则为 3。

winx = 3
winy = 3
## 视窗大小

num_filters = 2
## 计算 2 项特征
```

【卷积】是指：组固定的权重和不同窗口内数据做内积。卷积神经层的特点是：

（一）局部关联，每个神经元看做一个 filter 过滤者。

（二）窗口（receptive field）滑动，让 filter 对局部数据计算。

（三）参数共享，包括：深度（depth）、步长（stride）和填充值（zero-padding）。

（四）逐步计算，该机制假定每个神经元连接数据窗的权重是固定的，这样每个神经元只关注一个特性，使得需要估算的权重个数减少。

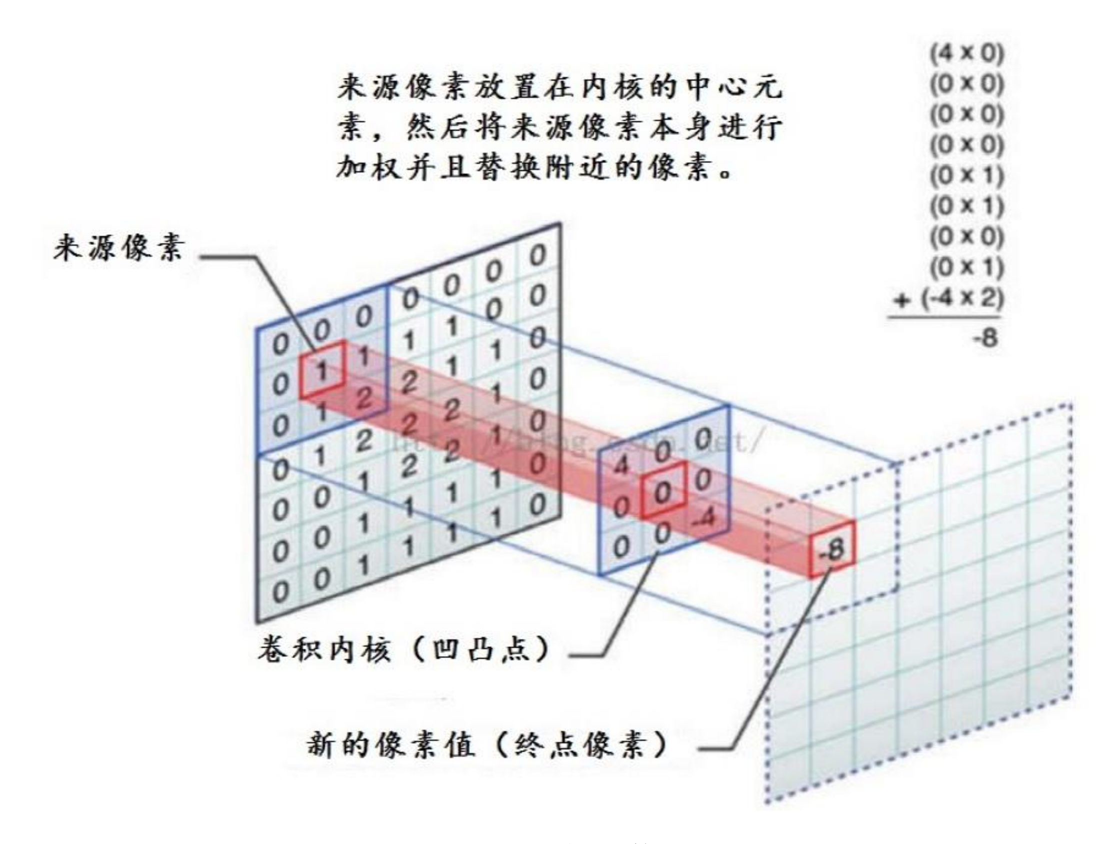


图 7.96 卷积计算层

为了对于卷积有所理解，我们可以阅读并且执行以下代码：

```
#卷积
W1 = tf.Variable(tf.truncated_normal([winx, winy,1, num_filters],
stddev=1./math.sqrt(winx*winy)))
## 1 代表 Channel 而视窗大小应与权重大小匹配。

b1 = tf.Variable(tf.constant(0.1,shape=[num_filters]))
xw = tf.nn.conv2d(x_im, W1, strides=[1, 1, 1, 1], padding='SAME')
##看到 x_im 是刚才定义 3x3 的卷积，此处一次一个像素。
## padding = SAME 是指即使超过输入图像的边界，滑动视窗仍会运行。

h1 = tf.nn.relu(xw + b1)
## 将卷积输出传递给激活函数 relu（修正线性单元函数），使得任何负输入都设置为零，
而正输入不变。

sess.run(tf.initialize_all_variables())
## 把对话初始化。
```

激励层是把卷积层输出的结果，进行非线性映射，主要包括：Sigmoid、Tanh(双曲正切)、ReLU、Leaky ReLU 和 ELU 几种情况。

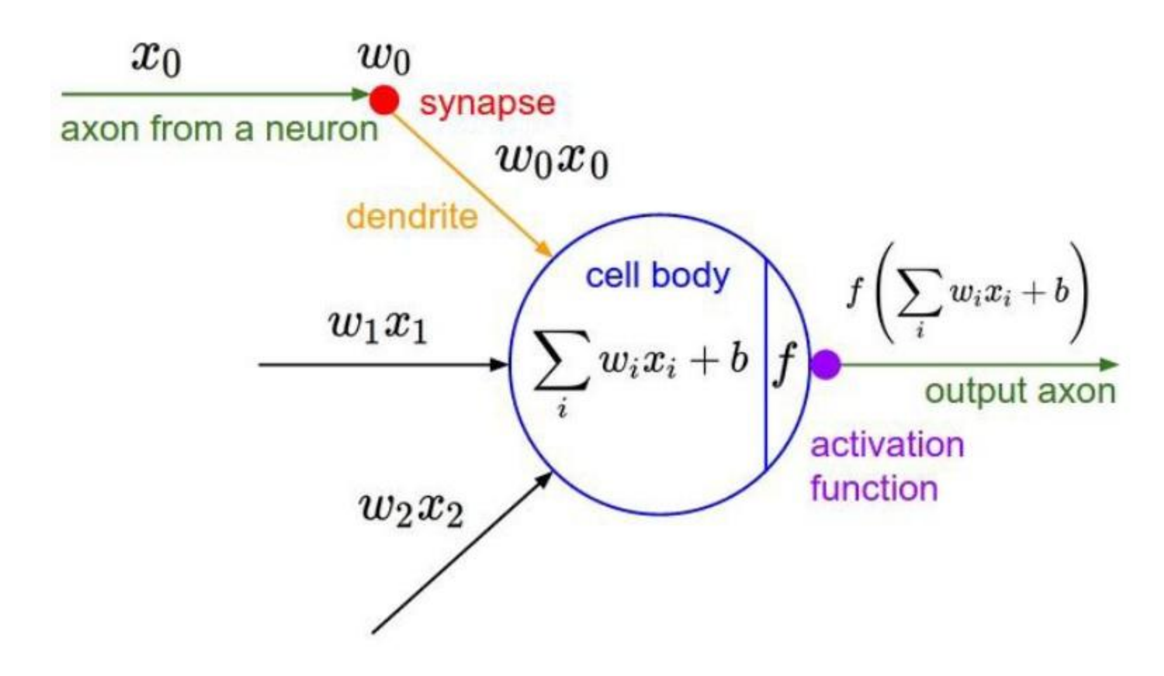


图 7.97 激励层的示意图

为了对于激励层有所理解，我们可以执行以下代码，在后面图 9.100 那里看到图形输出的（我截图的）汇总：

```
# 为了之后窥探卷积层的内部。
H = h1.eval(feed_dict = {x: image})

import matplotlib.pyplot as plt
plt.ion()

# 这里出现的是初始网络
plt.matshow(image[0])
plt.colorbar()

#第一个卷积网络的初始特征
plt.matshow(H[0,:,:,0])
plt.colorbar()

#第二个卷积网络的初始特征
plt.matshow(H[0,:,:,0])
plt.colorbar()
```

在众多激励层的类型中，最常讨论的是 Sigmoid 但是其它种类的激励层，也有各自的优缺点。例如 ReLU（The Rectified Linear Unit/修正线性单元）收敛快，求梯度简单，较脆弱；而 Leaky ReLU 则是不会 “饱和” （计算不了宕机），计算也快。至于 ELU 具备 ReLU 的优点，输出均值趋于零，但因为是指数，计算量略大 。

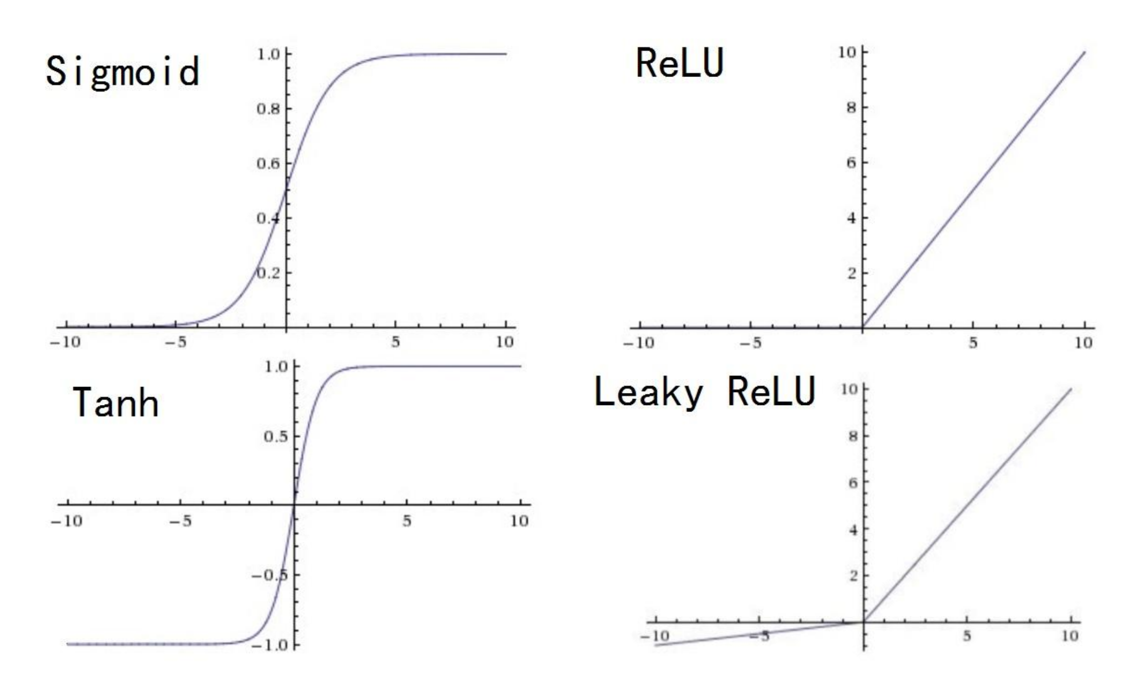


图 7.98 激励层的比较

池化层夹在连续的卷积层中间，目的是压缩数据和参数的量，减小过度拟合。全连接层（FC layer）通常在卷积神经网络尾部 ，两层之间所有神经元都有权重连接。

一般 CNN 结构依次为 ：

INPUT

[[CONV -> RELU]*N -> POOL?]*M

[FC -> RELU]*K

FC

图 7.99 池化层

对于池化层的理解，我们可以通过阅读以下代码。

```
#池化层
p1 = tf.nn.max_pool(h1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
p1_size = np.product([s.value for s in p1.get_shape()[1:]])
```

```
p1f = tf.reshape(p1, [-1, p1_size ])
## 需要选择-1 来选择合适形状，

P = p1.eval(feed_dict = {x: image})
## 整个窗口是 5x5 而最佳表示是每个子窗口 2x2 的形式。

plt.matshow(P[0,:,:,0])
     plt.colorbar()
```

我们刚才运行代码，经过【初始网络、第一个卷积、第二个卷积、池化层】等四层产生的数据特征的比较，如图 7.100 所示。

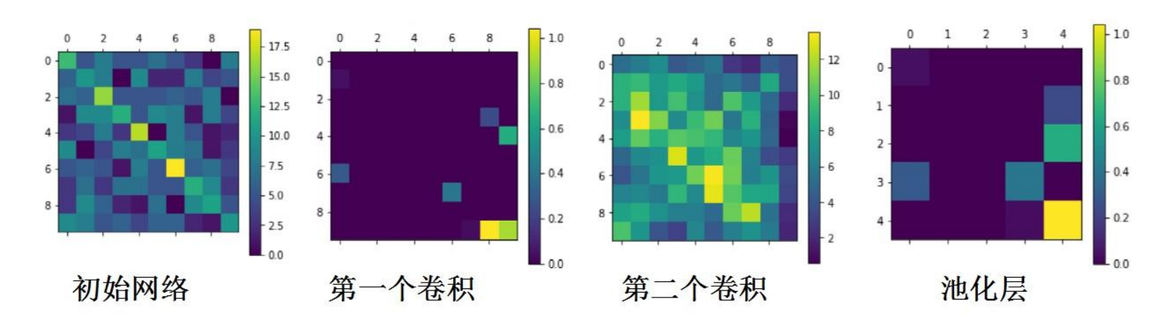


图 7.100 数据特征的比较

至此，我们构建了卷积神经网络的基本内容，但是还不是一个完整的卷积神经网络。类似前面第三部分那样，当初我们构建了神经网络的神经元，帮助我们理解神经网络是什么以及它的关键技术，但拿还不是神经网络，后来补充了许多内容之后，在 9.2.4 才算是真正实现第一个神经网络。那么，这里也是一样。接下来我们开展 15 项内容，真正把卷积神经网络进行到底。

为了便于理解和掌握，我们归纳 15 项任务为五个部分：

（一）准备环境、读取数据、转换数据；

（二）拆分数据、重塑数据；

（三）卷积层、密集层、输出层、模型化

（四）设计模型和精确度、训练数据

（五）绘制精确度的曲线图、查看混淆矩阵、检查权重、存储与调用权重

我们需要关注的是（二）拆分数据、重塑数据；以及（三）卷积层、密集层、输出层、模型化。与本节之前的第一部分和第二部分有所不同，因为这里对于数据标签化以及卷积网络的层级比较完整了。此外，与第二部分比较，我们把第一部分最后一部分加了进来：存储与调用权重。

第一部分：准备环境、读取数据、转换数据。

```
# 设置环境
import tensorflow as tf
import numpy as np
import math
try:
    from tqdm import tqdm
    except ImportError:
        def tqdm(x, *args, **kwargs):
            return x
```

```
# 设置随机种子
np.random.seed(0)

# 导入数据
data = np.load('data_with_labels.npz')
     train = data['arr_0']/255.
     labels = data['arr_1']

# 查看数据
print(train[0])
print(labels[0])

# 转换数据
import matplotlib.pyplot as plt
     plt.ion()
def to_onehot(labels,nclasses = 5):
         outlabels = np.zeros((len(labels),nclasses))
         for i,l in enumerate(labels):
             outlabels[i,l] = 1
         return outlabels
onehot = to_onehot(labels)
```

第二部分：拆分数据、重塑数据；

```
# 拆分数据
indices = np.random.permutation(train.shape[0])
valid_cnt = int(train.shape[0] * 0.1)
test_idx, training_idx = indices[:valid_cnt], indices[valid_cnt:]
test, train = train[test_idx,:], train[training_idx,:]
onehot_test, onehot_train = onehot[test_idx,:],onehot[training_idx,:]

sess = tf.InteractiveSession()

# 重塑数据
x = tf.placeholder("float", [None, 36, 36])
x_im = tf.reshape(x, [-1,36,36,1])
y_ = tf.placeholder("float", [None,5])
```

```
In [8]: import matplotlib.pyplot as plt
   ...: plt.ion()
   ...:
   ...: def to_onehot(labels,nclasses = 5):
   ...:     outlabels = np.zeros((len(labels),nclasses))
   ...:     for i,l in enumerate(labels):
   ...:         outlabels[i,l] = 1
   ...:     return outlabels
   ...:
   ...:
   ...: onehot = to_onehot(labels)

In [9]: indices = np.random.permutation(train.shape[0])
   ...: valid_cnt = int(train.shape[0] * 0.1)
   ...: test_idx, training_idx = indices[:valid_cnt], indices[valid_cnt:]
   ...: test, train = train[test_idx,:], train[training_idx,:]
   ...: onehot_test, onehot_train = onehot[test_idx,:],onehot[training_idx,:]

In [10]: sess = tf.InteractiveSession()
C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client\session.py:1702:
UserWarning: An interactive session is already active. This can cause out-of-memory errors in
some cases. You must explicitly call `InteractiveSession.close()` to release resources held
by the other session(s).
  warnings.warn('An interactive session is already active. This can '

In [11]: x = tf.placeholder("float", [None, 36, 36])
    ...: x_im = tf.reshape(x, [-1,36,36,1])
    ...: y_ = tf.placeholder("float", [None,5])
```

图 7.100 拆分数据、重塑数据

第三部分：卷积层、密集层、输出层、模型化

```
# 卷积层
num_filters = 4
winx = 5
winy = 5
W1 = tf.Variable(tf.truncated_normal( [winx, winy, 1 , num_filters],
stddev=1./math.sqrt(winx*winy)))
b1 = tf.Variable(tf.constant(0.1, shape=[num_filters]))
xw = tf.nn.conv2d(x_im, W1, strides=[1, 1, 1, 1], padding='SAME')
h1 = tf.nn.relu(xw + b1)
p1 = tf.nn.max_pool(h1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

# 密集层
p1_size = np.product([s.value for s in p1.get_shape()[1:]])
p1f = tf.reshape(p1, [-1, p1_size ])

num_hidden = 32
W2 = tf.Variable(tf.truncated_normal([p1_size, num_hidden],
stddev=2./math.sqrt(p1_size)))
b2 = tf.Variable(tf.constant(0.2, shape=[num_hidden]))
h2 = tf.nn.relu(tf.matmul(p1f,W2) + b2)

# 输出层
W3 = tf.Variable(tf.truncated_normal([num_hidden, 5],
stddev=1./math.sqrt(num_hidden)))
b3 = tf.Variable(tf.constant(0.1,shape=[5]))

keep_prob = tf.placeholder("float")
```

```
h2_drop = tf.nn.dropout(h2, keep_prob)

# 模型化
sess.run(tf.global_variables_initializer())
y = tf.nn.softmax(tf.matmul(h2_drop,W3) + b3)
```

第四部分：设计模型和精确度、训练数据

```
# 设计模型和精确度
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = y
+ 1e-50, labels = y_))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

# 训练数据
epochs = 5000
train_acc = np.zeros(epochs//10)
test_acc = np.zeros(epochs//10)
for i in tqdm(range(epochs), ascii=True):
     if i % 10 == 0:
         A = accuracy.eval(feed_dict={x: train,
             y_: onehot_train, keep_prob: 1.0})
         train_acc[i//10] = A
         A = accuracy.eval(feed_dict={x: test,
             y_: onehot_test, keep_prob: 1.0})
         test_acc[i//10] = A
     train_step.run(feed_dict={x: train,
         y_: onehot_train, keep_prob: 0.5})
```

第五部分：绘制精确度的曲线图、查看混淆矩阵、检查权重、存储与调用权重

```
# 绘制精确度的曲线图
plt.figure(figsize=(6, 6))
plt.plot(train_acc,'bo')
plt.plot(test_acc,'rx')

# 查看混淆矩阵
pred = np.argmax(y.eval(feed_dict={x: test, keep_prob: 1.0, y_: onehot_test}), axis
= 1)
conf = np.zeros([5,5])
for p,t in zip(pred,np.argmax(onehot_test,axis=1)):conf[t,p] += 1
plt.matshow(conf)
plt.colorbar()

# 检查权重
f, plts = plt.subplots(4)
for i in range(4):
     plts[i].matshow(
             W1.eval()[:,:,0,i])
```

```
plt.matshow(W3.eval())
plt.colorbar()

# 存储与调用权重
saver = tf.train.Saver()
saver.save(sess, "conv1.ckpt")
saver.restore(sess, "conv1.ckpt")
```

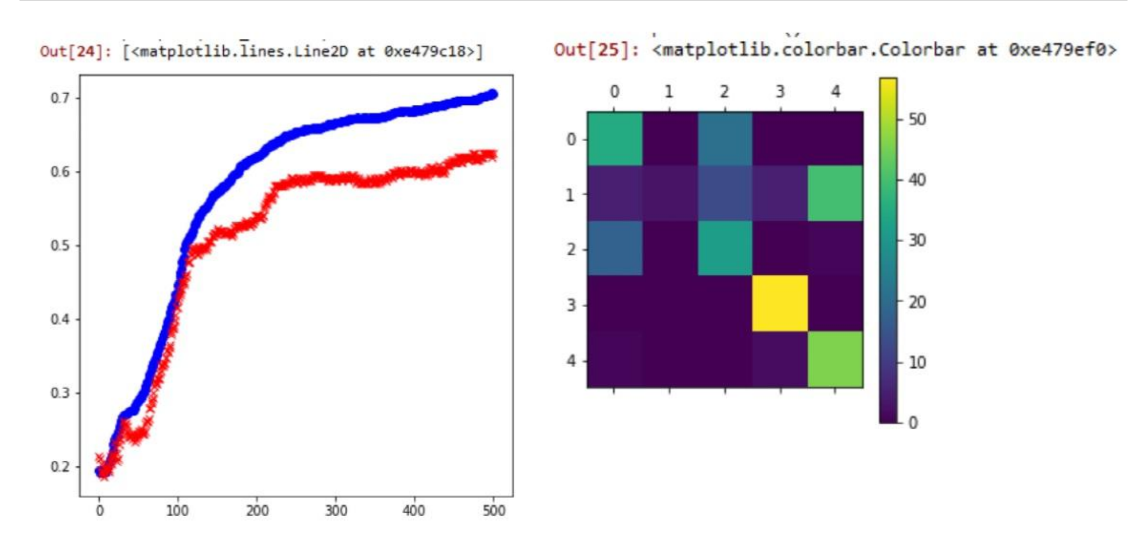
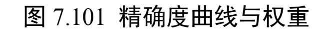


图 7.101 精确度曲线与权重



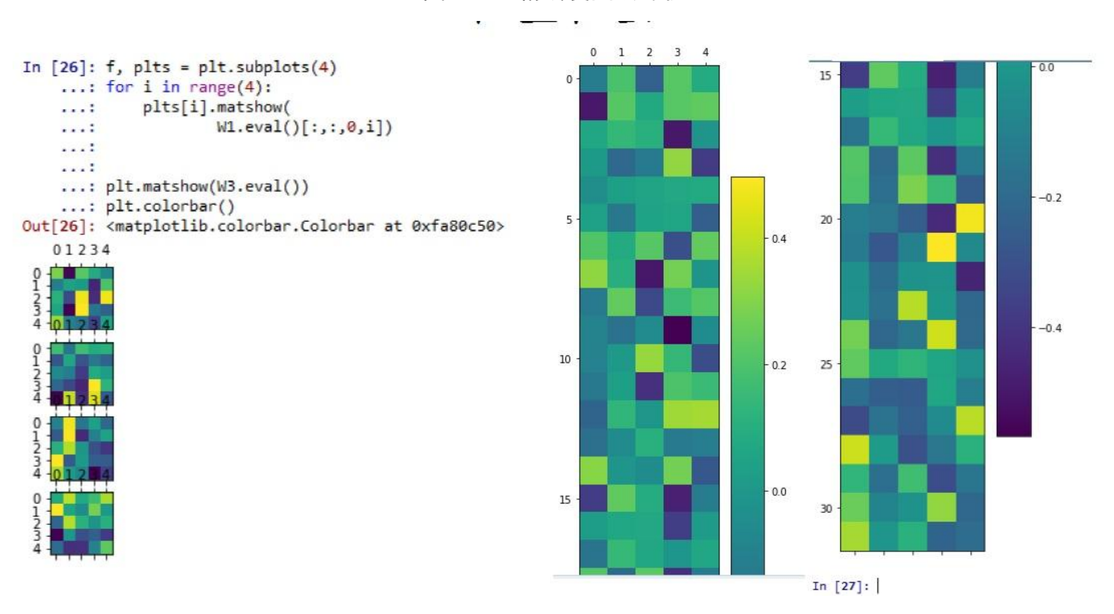


图 7.102 检查权重并且存储

三、更深层卷积神经网络

基于本节第一部分和第二部分的内容，我们可以进行【更深层卷积神经网络】的探讨，这一部分的内容较长，但是因为前面已经有了一些介绍，所以我们本节只需要代码和截图，

作为帮助理解的内容。

另外，在【更深层卷积神经网络】还有两个重要部分：模型评估和权重存取，我们把它们放到了第四部分讨论。此处我们只需关注前面一二部分和本部分的比较，即可。

此处，共有 17 项内容，为了不要眼花缭乱，我们归纳为五大部分，包括：

（一）准备环境、导入数据、转换数据；

（二）拆分数据、建立层级；

（三）卷积层之一、池化、卷积层之二、池化、密集层、输出层；

（四）建立度量标准、训练数据；

（五）查看精确度曲线、查看最终测试的混淆矩阵、检查输出权重、存储权重的信息、使用 Numpy 进行存储。

承上，我们在第二部分建立层级，做好分析的准备，接着在第三部分，增加了层级，是重点。在第五部分，增添了 Numpy 转换数组的小技巧，通过这个方式，我们又重新回到第二章数据处理的过程，这样算是在课程上有个小的闭环。

第一部分：准备环境、导入数据、转换数据。

```
# 准备环境
import tensorflow as tf
import numpy as np
import math
try:
     from tqdm import tqdm
     except ImportError:
     def tqdm(x, *args, **kwargs):
        return x
# 导入数据
np.random.seed(0)
data = np.load('data_with_labels.npz')
train = data['arr_0']/255.
labels = data['arr_1']

# 转换数据
import matplotlib.pyplot as plt
plt.ion()

def to_onehot(labels,nclasses = 5):
     outlabels = np.zeros((len(labels),nclasses))
     for i,l in enumerate(labels):
         outlabels[i,l] = 1
     return outlabels
## 转换为数组格式
onehot = to_onehot(labels)
```

第二部分：拆分数据、建立层级。

```
# 拆分数据
indices = np.random.permutation(train.shape[0])
valid_cnt = int(train.shape[0] * 0.1)
test_idx, training_idx = indices[:valid_cnt],\
                          indices[valid_cnt:]
```

```
test, train = train[test_idx,:],\
               train[training_idx,:]
onehot_test, onehot_train = onehot[test_idx,:],\
                         onehot[training_idx,:]
sess = tf.InteractiveSession()

# 建立层级
x = tf.placeholder("float", [None, 36, 36])
x_im = tf.reshape(x, [-1,36,36,1])
y_ = tf.placeholder("float", [None,5])
```

第三部分：卷积层之一、池化之一、卷积层之二、池化之二、密集层、输出层。

```
# 卷积层之一
num_filters1 = 16
winx1 = 3
winy1 = 3
W1 = tf.Variable(tf.truncated_normal([winx1, winy1, 1 ,
num_filters1],stddev=1./math.sqrt(winx1*winy1)))
b1 = tf.Variable(tf.constant(0.1, shape=[num_filters1]))

# 池化之一
xw = tf.nn.conv2d(x_im, W1,
                   strides=[1, 1, 1, 1],
                   padding='SAME')
h1 = tf.nn.relu(xw + b1)
## 卷积 5x5 并且带有 0 的边
p1 = tf.nn.max_pool(h1, ksize=[1, 2, 2, 1],
         strides=[1, 2, 2, 1], padding='VALID')
## 最大池化 2x2 并且不填充边

# 卷积层之二
num_filters2 = 4
winx2 = 3
winy2 = 3
W2 = tf.Variable(tf.truncated_normal(
    [winx2, winy2, num_filters1, num_filters2],
stddev=1./math.sqrt(winx2*winy2)))
b2 = tf.Variable(tf.constant(0.1, shape=[num_filters2]))

# 池化之二
p1w2 = tf.nn.conv2d(p1, W2, strides=[1, 1, 1, 1], padding='SAME')
h1 = tf.nn.relu(p1w2 + b2)
## 卷积 3x3 并且带有 0 的边
p2 = tf.nn.max_pool(h1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
## 最大池化 2x2 并且不填充边
p2_size = np.product([s.value for s in p2.get_shape()[1:]])
p2f = tf.reshape(p2, [-1, p2_size ])
## 扁平化卷积输出

# 密集层
```

```
num_hidden = 32
W3 = tf.Variable(tf.truncated_normal([p2_size, num_hidden],
stddev=2./math.sqrt(p2_size)))
b3 = tf.Variable(tf.constant(0.2, shape=[num_hidden]))
h3 = tf.nn.relu(tf.matmul(p2f,W3) + b3)

keep_prob = tf.placeholder("float")
h3_drop = tf.nn.dropout(h3, keep_prob)
## 退出训练

# 输出层。
W4 = tf.Variable(tf.truncated_normal([num_hidden, 5],
stddev=1./math.sqrt(num_hidden)))
b4 = tf.Variable(tf.constant(0.1,shape=[5]))

sess.run(tf.initialize_all_variables())
## 初始化
y = tf.nn.softmax(tf.matmul(h3_drop,W4) + b4)
## 定义模型
```

第四部分：建立度量标准、训练数据。

```

# 建立度量标准
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits( logits =
y + 1e-50, labels = y_))
## 交叉熵，用于度量两个概论分布之间的差异性信息。

train_step = tf.train.GradientDescentOptimizer( 0.01).minimize(cross_entropy)
## 梯度下降优化器，用于度量训练步骤。

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
## 预测标准，用于定义精确度。

accuracy = tf.reduce_mean(tf.cast( correct_prediction, "float"))
## 定义精确度。

# 训练数据
epochs = 6000
train_acc = np.zeros(epochs//10)
test_acc = np.zeros(epochs//10)
 for i in tqdm(range(epochs), ascii=True):
     if i % 10 == 0:
         A = accuracy.eval(feed_dict={x: train, y_: onehot_train, keep_prob: 1.0})
         train_acc[i//10] = A
         A = accuracy.eval(feed_dict={x: test, y_: onehot_test, keep_prob: 1.0})
         test_acc[i//10] = A
     train_step.run(feed_dict={x: train, y_: onehot_train, keep_prob: 0.5})
```

```
In [60]: epochs = 6000

In [61]: train_acc = np.zeros(epochs//10)

In [62]: test_acc = np.zeros(epochs//10)

In [63]: for i in tqdm(range(epochs), ascii=True):
    ...:     if i % 10 == 0:
    ...:         A = accuracy.eval(feed_dict={x: train, y_:
onehot_train, keep_prob: 1.0})
    ...:         train_acc[i//10] = A
    ...:         A = accuracy.eval(feed_dict={x: test, y_: onehot_test,
keep_prob: 1.0})
    ...:         test_acc[i//10] = A
    ...:     train_step.run(feed_dict={x: train, y_: onehot_train,
keep_prob: 0.5})
  7%|6          | 413/6000 [15:17<3:30:18,  2.26s/it]
```

• 计算时间大约一小时，所以深度学习是需要大数据体系或者高性能计算提供计算资源的。

图 7.103 更深层卷积神经网络训练耗费计算资源

第五部分：查看精确度曲线、查看最终测试的混淆矩阵、检查输出权重、存储权重的信息、使用 Numpy 进行存储。

```
# 绘制精确度曲线
plt.figure(figsize=(6, 6))
plt.plot(train_acc,'bo')
plt.plot(test_acc,'rx')

# 查看最终测试的混淆矩阵
pred = np.argmax(y.eval(feed_dict={x: test, keep_prob: 1.0, y_: onehot_test}), axis
= 1)
conf = np.zeros([5,5])
for p,t in zip(pred,np.argmax(onehot_test, axis=1)): conf[t,p] += 1
plt.matshow(conf)
plt.colorbar()

# 检查输出权重
f, plts = plt.subplots(4,4)
for i in range(16):
     plts[i//4,i%4].matshow(W1.eval()[:,:,0,i],
             cmap = plt.cm.gray_r)
plt.matshow(W4.eval().T)
plt.colorbar()

# 存储权重的信息
saver = tf.train.Saver()
saver.save(sess, "conv2a.ckpt")
saver.restore(sess, "conv2a.ckpt")
##这样（便于下次调用所存储的权重信息）只需要一个指令。
```

```
In [64]: plt.figure(figsize=(6, 6))
    ...: plt.plot(train_acc,'bo')
    ...: plt.plot(test_acc,'rx')
Out[64]: [<matplotlib.lines.Line2D at 0xe762588>]
```

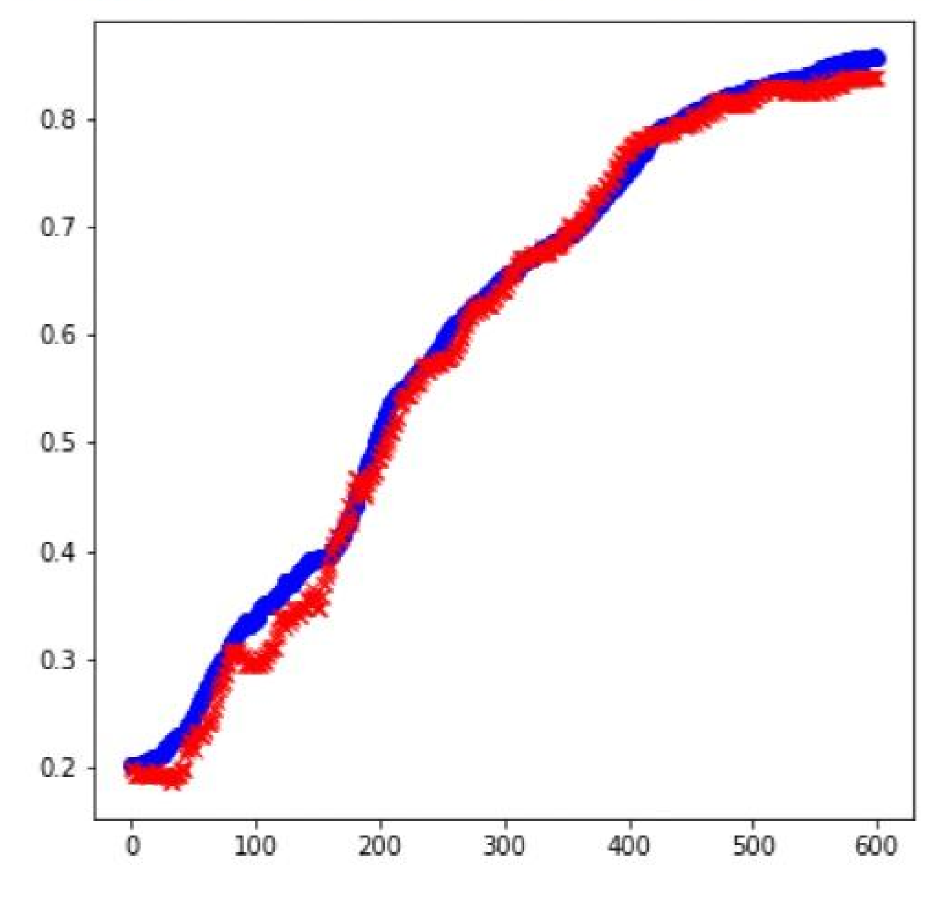


图 9.104 精确度曲线

```
In [65]: pred = np.argmax(y.eval(feed_dict={x: test, keep_prob: 1.0,
y_: onehot_test}), axis = 1)

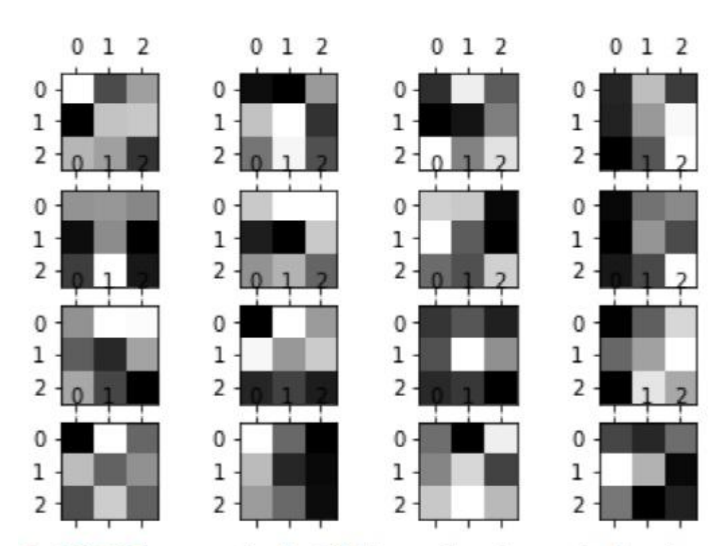
In [66]: conf = np.zeros([5,5])

In [67]: for p,t in zip(pred,np.argmax(onehot_test, axis=1)): conf[t,p]
+= 1

In [68]: plt.matshow(conf)
    ...: plt.colorbar()
Out[68]: <matplotlib.colorbar.Colorbar at 0xeae2048>
```

图 9.105 最终测试的混淆矩阵

```
In [69]: f, plts = plt.subplots(4,4)
    ...: for i in range(16):
    ...:     plts[i//4,i%4].matshow(W1.eval()[:,:,0,i],
    ...:                cmap = plt.cm.gray_r)
```



```
Out[70]: <matplotlib.colorbar.Colorbar at 0xed6f7b8>
```

图 7.106 输出权重

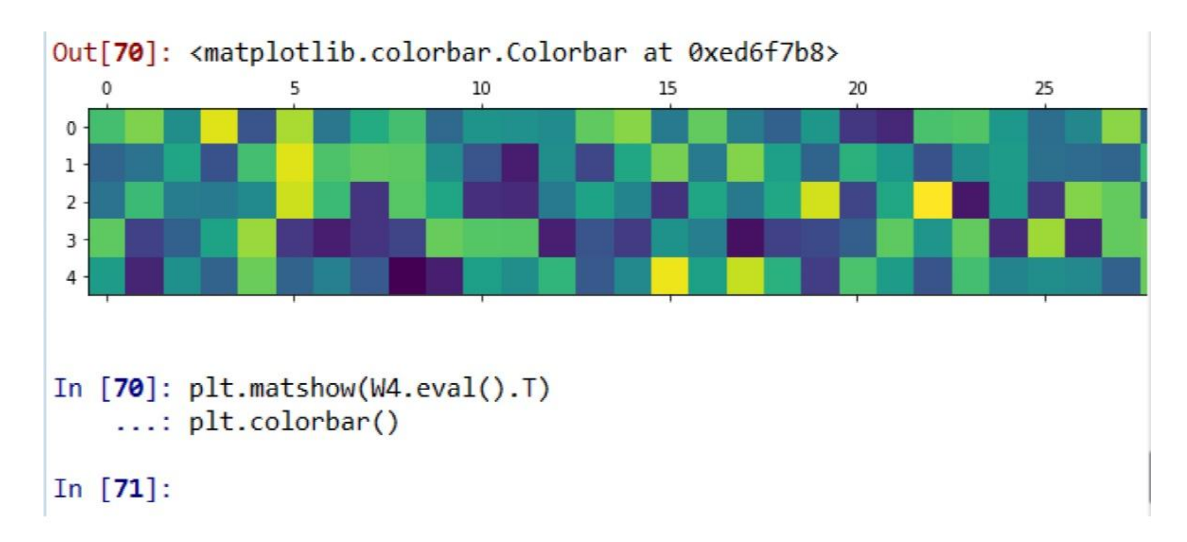


图 7.107 输出权重（2）

最后，我们考虑如何使用 Numpy 进行存储。

```
def save_all(name = ’conv2’):
    np.savez_compressed(name, W1.eval(),
            b1.eval(), W2.eval(), b2.eval(),
            W3.eval(), b3.eval(), W4.eval(),
            b4.eval())
save_all()
```

如何采用这种方式存储权重信息？我们需要定义【数组】再接着导入 numpy 的文件。

```
def load_all(name = ’conv2.npz’):
     data = np.load(name)
     sess.run(W1.assign(data[’arr_0’]))
     sess.run(b1.assign(data[’arr_1’]))
     sess.run(W2.assign(data[’arr_2’]))
     sess.run(b2.assign(data[’arr_3’]))
     sess.run(W3.assign(data[’arr_4’]))
     sess.run(b3.assign(data[’arr_5’]))
     sess.run(W4.assign(data[’arr_6’]))
     sess.run(b4.assign(data[’arr_7’]))
load_all()
```

```
In [74]: def save_all(name = 'conv2'):
    ...:     np.savez_compressed(name, W1.eval(),
    ...:             b1.eval(), W2.eval(), b2.eval(),
    ...:             W3.eval(), b3.eval(), W4.eval(),
    ...:             b4.eval())

In [75]: save_all()

In [76]: def load_all(name = 'conv2.npz'):
    ...:     data = np.load(name)
    ...:     sess.run(W1.assign(data['arr_0']))
    ...:     sess.run(b1.assign(data['arr_1']))
    ...:     sess.run(W2.assign(data['arr_2']))
    ...:     sess.run(b2.assign(data['arr_3']))
    ...:     sess.run(W3.assign(data['arr_4']))
    ...:     sess.run(b3.assign(data['arr_5']))
    ...:     sess.run(W4.assign(data['arr_6']))
    ...:     sess.run(b4.assign(data['arr_7']))

In [77]: load_all()
```

图 7.108 以数组方式存储信息

更深层卷积神经网络训练耗费计算资源（图 7.103），在个人计算机上，虽然仅有 5000 笔数据（均以数组格式存储的图片像素），但是仅仅因为两层卷积层和两层池化层的原因，计算时间大约一小时。所以，有此经验来看，深度学习是需要大数据体系（第八章）或者高性能计算提供计算资源的。

四、抽取权重与模型评估

因为前面第三部分已经训练好了模型，我们也把权重值存储起来了，所以现在要做的事情，就是把权重抽取出来训练模型，并且进行模型评估。

第一步：定义卷积学习、密集连接层、训练函数；

第二步：使用泛型估计器；

第三步：简化精确度。

```
# 定义卷积学习
def conv_learn(X, y, mode):
    X = tf.reshape(X, [-1, 36, 36, 1])
    y = tf.one_hot(tf.cast(y, tf.int32), 5, 1, 0)
## 确保图像是二维形式
## 热编码
with tf.variable_scope('conv_layer'):
## 卷积层为每个 5x5 补丁计算四个内核
 h1 = layers.convolution2d(X, num_outputs=4, kernel_size=[5, 5],
```

```
activation_fn=tf.nn.relu)
## 这些 5x5 的卷积在边上带有 0 填充（pad）
 p1 = tf.nn.max_pool(h1, ksize=[1, 2, 2, 1], trides=[1, 2, 2, 1], padding='VALID')
## 在 2x2 最大池化层不带 0 填充

# 密集连接层
p1_size = np.product([s.value for s in p1.get_shape()[1:]])
## 为密集层（dense layer）扁平化卷积输出层(conv output)
p1f = tf.reshape(p1, [-1, p1_size ])
h_fc1 = layers.fully_connected(p1f, 5, activation_fn=tf.nn.relu)
## 密集连接层（densely connected layer）有 32 个神经元和 dropout 层
drop = layers.dropout(h_fc1, keep_prob=0.5,  is_training=mode ==
tf.contrib.learn.ModeKeys.TRAIN)
logits = layers.fully_connected(drop, 5, activation_fn=None)
loss = tf.losses.softmax_cross_entropy(y, logits)

# 手动设置训练函数
train_op = layers.optimize_loss(
         loss,
         tf.contrib.framework.get_global_step(),
         optimizer='Adam',
         learning_rate=0.01)
     return tf.argmax(logits, 1), loss, train_op
```

结果：

```
In [53]: def conv_learn(X, y, mode):
    ...:     X = tf.reshape(X, [-1, 36, 36, 1])
    ...:     y = tf.one_hot(tf.cast(y, tf.int32), 5, 1, 0)
    ...:
    ...:     with tf.variable_scope('conv_layer'):
    ...:         h1 = layers.convolution2d(X, num_outputs=4,
    ...:                 kernel_size=[5, 5],
    ...:                 activation_fn=tf.nn.relu)
    ...:         p1 = tf.nn.max_pool(h1, ksize=[1, 2, 2, 1],
    ...:                 strides=[1, 2, 2, 1], padding='VALID')
    ...:     p1_size = np.product([s.value for s in p1.get_shape()[1:]])
    ...:     p1f = tf.reshape(p1, [-1, p1_size ])
    ...:     h_fc1 = layers.fully_connected(p1f,5, activation_fn=tf.nn.relu
    ...:     drop = layers.dropout(h_fc1, keep_prob=0.5, is_training=mode =
tf.contrib.learn.ModeKeys.TRAIN)
    ...:     logits = layers.fully_connected(drop, 5, activation_fn=None)
    ...:     loss = tf.losses.softmax_cross_entropy(y, logits)
    ...:     train_op = layers.optimize_loss(
    ...:         loss,
    ...:         tf.contrib.framework.get_global_step(),
    ...:         optimizer='Adam',
    ...:         learning_rate=0.01)
    ...:     return tf.argmax(logits, 1), loss, train_op
    ...:
```

图 7.109 定义卷积学习、密集连接层、训练函数

```
# 在函数中使用泛型估计器（generic estimator）
classifier = estimator.SKCompat(
          learn.Estimator(
          model_fn=conv_learn))

classifier.fit(train,train_labels,
                 steps=1024,
                 batch_size=32)
```

结果：

```
In [54]: classifier = estimator.SKCompat(learn.Estimator(model_fn=conv_lear
   ...: classifier.fit(train,train_labels, steps=1024, batch_size=32)
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\user
\AppData\Local\Temp\tmpvm3tkp_g
INFO:tensorflow:Using config: {'_task_type': None, '_task_id': 0,
'_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object
0x0000000013EE8208>, '_master': '', '_num_ps_replicas': 0,
'_num_worker_replicas': 0, '_environment': 'local', '_is_chief': True,
'_evaluation_master': '', '_train_distribute': None, '_eval_distribute': No
'_device_fn': None, '_tf_config': gpu_options {
  per_process_gpu_memory_fraction: 1.0
}
, '_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_s
600, '_log_step_count_steps': 100, '_protocol': None, '_session_config': No
'_save_checkpoints_steps': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_model_dir': 'C:\\Users\\user\\Ap
\\Local\\Temp\\tmpvm3tkp_g'}
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorfl
\python\keras\layers\core.py:143: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be remo
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
INFO:tensorflow:global_step/sec: 22.1177
INFO:tensorflow:loss = 1.4089344, step = 101 (4.523 sec)
INFO:tensorflow:global_step/sec: 37.1864
INFO:tensorflow:loss = 0.87586397, step = 201 (2.689 sec)
INFO:tensorflow:global_step/sec: 42.5508
INFO:tensorflow:loss = 0.7943554, step = 301 (2.350 sec)
INFO:tensorflow:global_step/sec: 47.0119
INFO:tensorflow:loss = 0.80787396, step = 401 (2.126 sec)
INFO:tensorflow:global_step/sec: 52.4354
INFO:tensorflow:loss = 0.78303826, step = 501 (1.907 sec)
INFO:tensorflow:global_step/sec: 52.518
INFO:tensorflow:loss = 0.9150031, step = 601 (1.904 sec)
INFO:tensorflow:global_step/sec: 51.7569
INFO:tensorflow:loss = 0.8237965, step = 701 (1.932 sec)
INFO:tensorflow:global_step/sec: 51.2791
INFO:tensorflow:loss = 0.6588527, step = 801 (1.950 sec)
INFO:tensorflow:global_step/sec: 52.0262
INFO:tensorflow:loss = 0.74859035, step = 901 (1.922 sec)
INFO:tensorflow:global_step/sec: 51.7837
INFO:tensorflow:loss = 0.9293049, step = 1001 (1.931 sec)
INFO:tensorflow:Saving checkpoints for 1024 into C:\Users\user\AppData\Local
\Temp\tmpvm3tkp_g\model.ckpt.
INFO:tensorflow:Loss for final step: 0.8850194.
Out[54]: SKCompat()
```

图 7.110 使用泛型估计器

```
# 简化精确度
metrics.accuracy_score(test_labels,classifier.predict(test))
```

结果：

```
In [55]: metrics.accuracy_score(test_labels,classifier.predict(test))
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\user\AppData\Local\Temp
\tmpvm3tkp_g\model.ckpt-1024
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Out[55]: 0.8530465949820788
```

图 7.111 简化精确度

抽取权重之后的模型评估，所需工作，仅有三步：

（一）导入数据；

（二）重塑数据；

（三）查看网络。

```
# 导入数据
import tensorflow as tf
import numpy as np
try:
    from tqdm import tqdm
  except ImportError:
```

```
    def tqdm(x, *args, **kwargs):
        return x
data = np.load('data_with_labels.npz')
train = data['arr_0']/255.
labels = data['arr_1']
print(train[0])
print(labels[0])
```

结果：

```
In [1]: import tensorflow as tf

In [2]: import tensorflow.contrib.layers as layers

In [3]: import numpy as np

In [4]: try:
   ...:     from tqdm import tqdm
   ...: except ImportError:
   ...:     def tqdm(x, *args, **kwargs):
   ...:         return x
   ...:

In [5]: data = np.load('data_with_labels.npz')
   ...: train = data['arr_0']/255.
   ...: labels = data['arr_1']

In [6]:
```

图 7.112 导入数据

```
# 重塑数据：重塑是指制作一个大型数组，然后可视化。
import matplotlib.pyplot as plt
plt.ion()
all_letters = np.zeros([5*36,62*36])
for font in range(5):
    for letter in range(62):
    all_letters[font*36:(font+1)*36, letter*36:(letter+1)*36] = train[9*(font*62
+ letter)]
plt.pcolormesh(all_letters, cmap=plt.cm.gray)
```

结果：

```
In [8]: all_letters = np.zeros([5*36,62*36])
   ...: for font in range(5):
   ...:     for letter in range(62):
   ...:         all_letters[font*36:(font+1)*36, letter*36:(letter+1)*36] =
train[9*(font*62 + letter)]
   ...: plt.pcolormesh(all_letters, cmap=plt.cm.gray)
Out[8]: <matplotlib.collections.QuadMesh at 0xe4c8f28>
```

图 7.113 重塑数据

```
# 查看网络
f, plts = plt.subplots(3,3, sharex=True, sharey=True)
for i in range(3):
    for j in range(3):
        plts[i,j].pcolor(train[i + 3*j])
```

结果：

```
In [9]: f, plts = plt.subplots(3,3, sharex=True, sharey=True)
   ...: for i in range(3):
   ...:     for j in range(3):
   ...:         plts[i,j].pcolor(train[i + 3*j])
   ...:
```

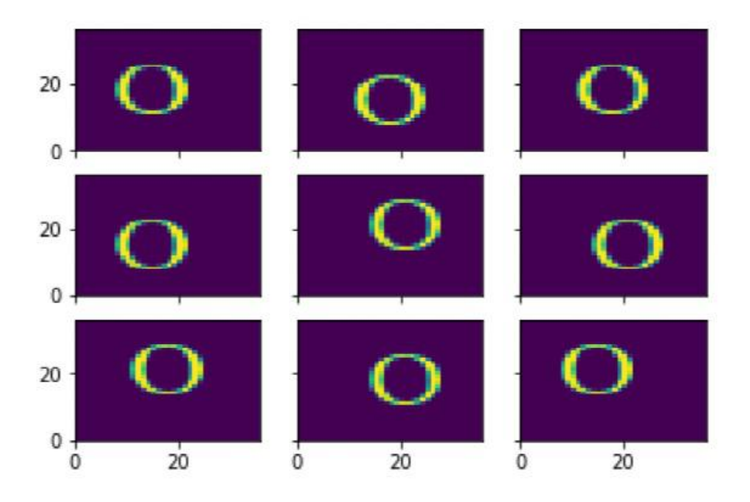


图 7.114 查看网络

本章小结

本章介绍了深度学习的知识，因为时间和篇幅有限，我们挑选了比较简单直白的内容予以讲解介绍。主线来自大数据（第六章）的分析能力的局限以及机器学习（第七章）的存储计算局限，弥补两者弱点结合两者优点，进行新的建模工作（第六章），将会产生新的算法（第三章）以及新的数据（第二章）的情况。

本章主要讲述了以下内容：

（1）深度学习的发展历史是过一个不断解决数据问题和计算问题的过程。

（2）掌握工具之一的 Tensorflow 的设计思路、解题思路和基本操作。

（3）再次理解罗吉斯回归的内容以及其它在此基础之上的理论、模型和算法的变化和进步。

（4）再次理解神经网络的内容以及使用 Tensorflow 进行操作的代码指令及其。

（5）学习罗吉斯回归、神经神经、深层神经网络、卷积神经网络的连贯关系。

（6）深刻掌握模型求取权重以及算法需要评估的意义。

（7）具体操作 TensorFlow 进行上述内容的实践，并且能够解读 TF 的代码和指令及其修改方式。进一步能够阅读 Tensorflow 代码，以及思考如何应用在学科领域的科学研究上。最后，从第七章倒推第一章，完成从做中学的知识经验的体验上的一个总的闭环。

习　　题

一、单选题

1、深度学习的构建【无关】下列哪个模型：（___）

（A）最小二乘线性回归

（B）罗吉斯回归

（C）神经元

（D）Sigmoid 函数

2、深度学习【不包括】下列哪个：（___）

（A）卷积神经网络

（B）深度神经网络

（C）深度置信网络

（D）社会神经网络

二、多选题

1、在特征表示的细粒度上，数据处理的顺序是（___）

（1）结构化特征

（2）浅层特征表示

（3）特征抽取

（4）特征向量空间

2、在 TF 进行罗吉斯回归的顺序是（___）
（1）导入模块与数据
（2）绘制精确度曲线
（3）训练模型
（4）拆分数据为训练集和验证集

三、问答题

1、请简要说明机器学习与深度学习的相同点、相异点，以及彼此之间的关系？
7、请说明神经网络、深层神经网络以及卷积深层神经网络的相同点和相异点？

四、课后作业

根据上一章的研究设计，以及之前阅读、消化、吸收、理解他人的十篇科研成果的基础上，本周开始自己采集、整理、汇总、分析数据，可以用到从第一章到第九章的知识技能和软件工具甚至代码等。

五、作业答疑

不要紧张，我们只是作业，可以模仿照做，只要标出参考文献即可。如果是正式发表的出版物，则还要仔细考虑，在科研早期阶段，先不急于求成，要有什么重要成果发表，但是需要急于练习，旨在通过模仿他人工作以及课程交流学习，养成良好习惯，并且通过不断练习，积累初始的科研经验。

结 语

数据科学是一个交叉学科。其内核包括：统计学、计算机科学、人文艺术以及领域知识。其外延，应用在天文科学、地理科学、生命科学和心理科学等 STEM 学科，也在数字人文等公共政策研究上发挥作用。我们建立了一套贯穿 400 个训练模块的基础知识体系，用于 40 课时的通识教育。目前这份教材，提供其中 81 个模块，主要是为了《数据科学 R 与 Python 实践》（120500MGX003H）这门课程所编制的材料。

该课的授课目标是培养青年科学家，具备科学思维和数据素质。教师以科学社会主义介绍数据科学的发展和应用；组织学员们使用代码和数据，来掌握方法、工具和技能。

课程内容包括数据结构、数据类型、数据处理、数据分析、数据可视化、算法调试、机器学习、大数据应用、人工智能、数据共享和数据安全等。我们利用 R 和 Python 两种语言，予以演示和解释。学员们以“从做中学”的方式完成课堂练习和课后作业。

授课方式为从简单、进阶到应用，分为三级。第一级是知识点的记忆，第二级是数据集的操作，第三级是面对面的讨论。

所有学员都可在课堂上完成训练，获得第一级（分数 0~80）；学员完成课后作业，得到第二级（分数 81~90），少数学员经过鉴别通过考核，能够达到第三级（分数 91~99）。

本课程自 2018 年起开开课至今，均为春季课程，数据科学的进展快速，而且内容丰富，所以每年课程教材和内容都有大幅更新；自 2018 年至今，每年更新课程教材和内容：

- 主讲教师的简历 http://people.ucas.ac.cn/~alan
- 课程讲座的视频 https://i.youku.com/alanku

自 2018 年的 90 位学员，至 2020 年的 120 位学员，至 2022 年的 240 位学员，修习这门课程的人数越加庞大，所涉及的学科专业越加细分和分殊。授课教师应当在已有基础上，在同一课纲上，准备三种不同程度的教材、课件和内容。通过课程上的及时测验，了解和掌握学员的认识程度和能力，适当更换适合学员程度的教材。如此一来，相同内容的备课，就要增加三倍。而且，这不是简单地重复三遍，也不是在一份教材上进行增和删的工作，而是，既要把握主轴主线，又要按照不同程度予以重新编排和设计。

为什么？因为我国当前如此需要这类人才，如果耽误一分钟，就是耽误 240 分钟，而且就是科研的 240 分钟，如以 40 课时计算，就是 9600 小时的科研光阴。那么能够做多少事？反之，如果 40 小时的课程，能够提升诸多青年的诸多能力的话，那么，个人花上 800 天还是 240 天的时间，都不值一提。

个人力量不如集体。授课老师不断与其它老师互动，不断与学员互动，解决问题的过程中，吸取和消化那些原本没有考虑或者考虑不周的情况。令人欣喜的是，已有部分毕业学员，甚至结课之后，已经能够超越那次课程的内容，在“从做中学”中，达到自我实现和超越讲师的程度。这是最令人感到欣慰和兴奋的了。